BASICのテクニックがピクピクわかってくる新連載

BASICピクニック

#1 変体文字ツアー

ゲームプログラムを走らせるとときどき変な文字になることがある。この変体文字を作り出すかんたんなテクニックと仕組みをめぐるピクニック。

変体文字の秘密はVRAMのなかにある

ときどき「BASICを勉強しています」というお便りを見かけて、ちょっとちがうんじゃないかと思う。BASICは勉強するものじゃなく、思考シミュレーションを楽しむものだ。

BASICの基本ルール(命令の使い方)は、たぶんMファンの2ページぶんに入りきるくらいしかない。ただ、そのルールが、ぼくたちのふつうの暮らしとかけはなれているために、マニュアルは厚く、むずかしそうなものになってしまう。

でも、BASICのいいところは、ルールを少ししか知らなくてもいろんな遊び方ができるという点なのだ。BASICの文法をうろ覚えでも、ちょっとした仕組みを知れば、すぐに新しい遊びが発見できる。分厚いマニュアルは、辞書みたいに必要に応じて開くだけで十分だ。

辞書を持って、BASICの不思議な世界にピクニックに行こう。ぼくたちのまえにあるMSXは、わずかなプログラムだけでいろんなおもしろい現象を見せてくれるのだ。

```
0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[¥]^_`abcdefghijk
lmnopqrstuvwxyz
Ok
```

⬆SCREEN 1のごくふつうの文字 ⬇数行のプログラムでちょこちょこっとVRAMをいじくるだけでたとえば下のような変体文字になってしまう

```
0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[¥]^_`abcdefghijk
lmnopqrstuvwxyz
Ok
```

■変体文字の出生地

今回のピクニックは、ゲームプログラムを走らせたときなどに出てくる、太い文字、斜めの文字、絵になった文字、色つきの文字などの変体文字の秘密と作り方を探っていこう。

変体文字の出生地はVRAM(ブイラム)だ。VRAMとは、MSX本体のなかにある「画面」だと考えてほぼまちがいない。VRAMには、文字の形や色、表示されている位置、スプライトの形や色や位置のデータが記憶されていて、ディスプレイの画面はこのVRAMのデータを光の情報に変えたものにすぎない。

だから、VRAMのデータを直接いじると、それに応じて画面も変わる。では、どうすればVRAMを直接いじれるのか、どういじれば変体文字が出てくるのか。以下、次ページ。

BASIC　すべてのMSXにROMの形で内蔵されている思考シミュレーションソフト。正確にいうと、BASIC言語用の翻訳ソフト(インタプリタ)がROMになって内蔵されている。ほかのパソコンでも同様に内蔵されるか、ディスクなどで付属していることが多い。basic(基本的)という単語と同じつづりだが、じつは「Beginner's All-purpose Symbolic Instruction Code(ビギナーズ・オールパーポス・シンボリック・インストラクション・コード)」の頭文字を取ったものである。直訳すると、初心者向け汎用記号命令体系となるが、ようするに、計算やゲームや事務処理などいろいろ使えるやさしいプログラミング言語という意味だ。

VRAM　Video RAM(ビデオ・ラム＝画像用RAM)の略。MSXの本体にはメインRAMとVRAMの2タイプのRAMが入っている。メインRAMはたんにRAMともいい、プログラム自体や変数の内容などを記憶しているところ。それに対してVRAMは、名前のとおり、画面表示に関わるすべての情報を記憶・管理しているところである。メインRAMの内容はPEEK関数で読み出し、POKE命令で書きこむが、VRAMはそれぞれVPEEK関数、VPOKE命令でおこなう。VRAMの内容が変われば、それに応じて画面表示も変わる。

SCREEN1　MSX-BASICのテキストモードの1つ、およびそのモードにするための命令文。文字の表示がかんたんにでき、しかもSCREEN0とちがってスプライトが使えるため、BASICのゲームプログラムはこのスクリーンモードで作られることが多い。

※今回掲載のプログラムはすべてMSX(RAM8K)、MSX2/2+対応です。

アドレス　コンピュータで使うときは「番地」と訳す。MSXのなかのメモリ(ROM、RAM、VRAM)は、2進数8桁ぶん(これを1バイトという)を1単位とする記憶場所をたくさん持っていて、その1つ1つに番地、つまりアドレスがつけられている。この番地を頼りにして、必要なデータを読み出したり、書きこんだりする。

16進表記　16進数の形で数を書くこと。10～15をA～Fで表すため、1桁で0～15を表すことができる。頭に「&H」をつけて、たとえば&H0208と書かれ、「アンドエッチゼロニーゼロハチ」というふうに1桁ごとに読む。BASICでは、&Hのついた数は16進数として扱う。

2進表記　2進数の形で数を書くこと。1桁で0か1しか表さないので、10進数の10を表すにも4桁必要になる。メモリはじっさいには、この2進数の形で記憶しているので、2進表記で見たほうがわかりやすいデータも多い。右ページのパターンデータはその典型的な例だ。BASICでは、&Bのついた数は2進数として扱う。

OR　論理演算の1つ。10進数でこの演算の効果を見るのはむずかしいが、2進表記で見た場合は単純だ。演算の対象となる2つの数をいったん2進表記にして、その2つを重ねあわせただけで答えになる。右ページのプログラム行60で使われている。

D¥2　同じく右ページのプログラム行60の計算。¥は割り算の記号の一種で、ふつう使われる「／」とちがうところは、計算の対象となる数も答えも小数点以下を切り捨てておこなう点。VRAMのデータはかならず整数になるので、この記号を使うことが多い。ちなみに、ある2進数を2で割った答えは、もとの2進数を右に1桁ずらした形になる。また、4で割ると2桁ずれる(2で2回割ったのと同じ)し、8で割ると3桁ずれる(2で3回割ったのと同じ)。パターンをずらすときによく使われる。

SCREEN1のVRAMの構成

SCREEN文を実行するたびに、VRAMはそれぞれのモードに応じて割りふりされ、モード1では以下のようになる。今回のピクニックではパターンジェネレータテーブルとカラーテーブルだけをとりあげた。ほかの部分はまた別の機会に。

アドレス	内容	
&H0000～&H07FF	パターンジェネレータテーブル	
	未使用	
&H1800	パターン名称テーブル	
&H1B00		スプライト属性テーブル
&H2000		カラーテーブル
	未使用 (MSX2/2＋ではカラーテーブルのあとにパレットテーブルがある)	
&H3800～&H3FFF	スプライトパターンテーブル	

パターンジェネレータテーブルで遊ぼう

ここでは話をSCREEN1に限定して進めよう。

VRAMのなかには、文字の形を記憶している部分があり、そのエリアを「パターンジェネレータテーブル」という。文字の形は、このパターンジェネレータテーブル(以下PGTと略)にあるデータで決められているのだ。右ページの左右にならんでいるデータは、VRAMのPGTのアドレス(16進表記)とその中身(2進表記)だが、1と0がならんだ中身のほうを薄目で見てほしい。文字の形が見えてくるはずだ。ようするに、データの「1」の部分が文字の形になっているわけだ。

1つの文字は8×8ドットでできているが、その横1列8ドットぶんが2進数8桁(8ビット＝1バイト)に対応しているのがよくわかると思う。だから、1文字の形のデータは8バイト、つまり、8つのアドレスにまたがっているのだ。SCREEN1のVRAMでは、ある文字のパターンはそのキャラクタコード×8のアドレスから8バイトぶんのエリアに入っている。

下や右ページ中央付近にあるプログラムは、どれももとからVRAMにある文字のデータにかんたんな計算を加えることによって新しいデータを作り、それを同じ場所に書きこむという仕組みになっている。

各プログラムの計算の部分でどういうことをやっているのかは、ここではあまり説明できない。マニュアルやこれまでのファンダムの記事などで調べながら自分で考えてみてほしい。

パターンを二重にした乱視文字

下を太くしたコンピュータ文字

形を斜めにゆがめた斜体文字

太くして網かけしたネオン文字

左右を反転させた鏡文字

縦も横も太くした極太文字

いろいろな変体文字用プログラム

●乱視文字

```
10 SCREEN1:COLOR 15,4,7:WIDTH 29
20 ST=ASC("0"):LA=ASC("z")
30 FOR I=ST TO LA:PRINT CHR$(I);:NEXT
40 FOR I=ST*8 TO LA*8+7
50 D=VPEEK(I)
60 D=D AND D¥2 OR D¥4 OR D
70 VPOKE I,D
80 NEXT
```

●コンピュータ文字(行10～30は上と同じ)

```
40 FOR I=ST*8 TO LA*8+7
50 D=VPEEK(I)
60 D=D OR D¥(1-(IMOD8>3))
70 VPOKE I,D
80 NEXT
```

●斜体文字(行10～30は上と同じ)

```
40 FOR I=ST*8 TO LA*8+7
50 D=VPEEK(I)
60 D=D OR D¥2
70 VPOKE I,D¥VAL(MID$("84442211",IMOD8+1
,1))
80 NEXT
```

●ネオン文字(行10～30は上と同じ)

```
40 FOR I=ST*8 TO LA*8+7
50 D=VPEEK(I)
60 D=(D OR D¥2) AND 85*(IMOD2+1)
70 VPOKE I,D
80 NEXT
```

●鏡文字(行10～30は上と同じ)

```
40 FOR I=ST*8 TO LA*8+7
50 D=VPEEK(I)
60 DD=0:FOR J=0 TO 7:DD=DD-((D AND 2^J)>
0)*2^(7-J):NEXT
70 VPOKE I,DD
80 NEXT
```

●極太文字(行10～30は上と同じ)

```
40 FOR I=LA*8+7 TO ST*8 STEP -1
50 D=VPEEK(I)
60 IF (IAND7)=0 THEN D1=0 ELSE D1=VPEEK(
I-1):D1=D1 OR D1¥2
70 D=D OR D¥2 OR D1
80 VPOKE I,D
90 NEXTI
```

文字が太くなる仕組み

使用前(パターンジェネレータテーブルの一部)

VRAMのアドレス	パターン
&HØ1FØ(496)	11ØØØØØØ
&HØ1F1(497)	Ø11ØØØØØ
&HØ1F2(498)	ØØ11ØØØØ
&HØ1F3(499)	ØØØ11ØØØ
&HØ1F4(5ØØ)	ØØ11ØØØØ
&HØ1F5(5Ø1)	Ø11ØØØØØ
&HØ1F6(5Ø2)	11ØØØØØØ
&HØ1F7(5Ø3)	ØØØØØØØØ
&HØ1F8(5Ø4)	Ø111ØØØØ
&HØ1F9(5Ø5)	1ØØØ1ØØØ
&HØ1FA(5Ø6)	ØØØØ1ØØØ
&HØ1FB(5Ø7)	ØØØ1ØØØØ
&HØ1FC(5Ø8)	ØØ1ØØØØØ
&HØ1FD(5Ø9)	ØØØØØØØØ
&HØ1FE(51Ø)	ØØ1ØØØØØ
&HØ1FF(511)	ØØØØØØØØ
&HØ2ØØ(512)	Ø111ØØØØ
&HØ2Ø1(513)	1ØØØ1ØØØ
&HØ2Ø2(514)	ØØØØ1ØØØ
&HØ2Ø3(515)	Ø11Ø1ØØØ
&HØ2Ø4(516)	1Ø1Ø1ØØØ
&HØ2Ø5(517)	1Ø1Ø1ØØØ
&HØ2Ø6(518)	Ø111ØØØØ
&HØ2Ø7(519)	ØØØØØØØØ
&HØ2Ø8(52Ø)	ØØ1ØØØØØ
&HØ2Ø9(521)	Ø1Ø1ØØØØ
&HØ2ØA(522)	1ØØØ1ØØØ
&HØ2ØB(523)	1ØØØ1ØØØ
&HØ2ØC(524)	11111ØØØ
&HØ2ØD(525)	1ØØØ1ØØØ
&HØ2ØE(526)	1ØØØ1ØØØ
&HØ2ØF(527)	ØØØØØØØØ
&HØ21Ø(528)	1111ØØØØ
&HØ211(529)	Ø1ØØ1ØØØ
&HØ212(53Ø)	Ø1ØØ1ØØØ
&HØ213(531)	Ø111ØØØØ
&HØ214(532)	Ø1ØØ1ØØØ
&HØ215(533)	Ø1ØØ1ØØØ
&HØ216(534)	1111ØØØØ
&HØ217(535)	ØØØØØØØØ
&HØ218(536)	ØØ11ØØØØ
&HØ219(537)	Ø1ØØ1ØØØ
&HØ21A(538)	1ØØØØØØØ
&HØ21B(539)	1ØØØØØØØ
&HØ21C(54Ø)	1ØØØØØØØ
&HØ21D(541)	Ø1ØØ1ØØØ
&HØ21E(542)	ØØ11ØØØØ
&HØ21F(543)	ØØØØØØØØ

VRAMのアドレス 16進表記(カッコ内は10進表記)

文字の1ラインごとのパターンデータ 2進表記

数字のØから小文字のzまでを太くするプログラム

```
10 SCREEN 1:COLOR 15,4,7:WIDTH 29
20 ST=ASC("Ø"):LA=ASC("z")
30 FOR I=ST TO LA:PRINT CHR$(I);:NEXT
40 FOR I=ST*8 TO LA*8+7
50 D=VPEEK(I)
60 D=D OR D¥2
70 VPOKE I,D
80 NEXT
```

ST、LAはそれぞれここで扱う文字群の最初と最後のキャラクタコード。DはVRAMから読みこんだデータ、およびVRAMに書きこむデータ。行10から30で0～zを表示し、行40から行80で下のほうで説明してあるような作業をくりかえしている。

上のプログラムを実行するとこんなことが起こる

```
Ø123456789:;<=>?@ABCDEFGHIJKL
MNOPQRSTUVWXYZ[¥]^_`abcdefghi
jklmnopqrstuvwxyz
Ok
```

使用後

VRAMのアドレス	パターン
&HØ1FØ(496)	111ØØØØØ
&HØ1F1(497)	Ø111ØØØØ
&HØ1F2(498)	ØØ111ØØØ
&HØ1F3(499)	ØØØ111ØØ
&HØ1F4(5ØØ)	ØØ111ØØØ
&HØ1F5(5Ø1)	Ø111ØØØØ
&HØ1F6(5Ø2)	111ØØØØØ
&HØ1F7(5Ø3)	ØØØØØØØØ
&HØ1F8(5Ø4)	Ø1111ØØØ
&HØ1F9(5Ø5)	11ØØ11ØØ
&HØ1FA(5Ø6)	ØØØØ11ØØ
&HØ1FB(5Ø7)	ØØØ11ØØØ
&HØ1FC(5Ø8)	ØØ11ØØØØ
&HØ1FD(5Ø9)	ØØØØØØØØ
&HØ1FE(51Ø)	ØØ11ØØØØ
&HØ1FF(511)	ØØØØØØØØ
&HØ2ØØ(512)	Ø1111ØØØ
&HØ2Ø1(513)	11ØØ11ØØ
&HØ2Ø2(514)	ØØØØ11ØØ
&HØ2Ø3(515)	Ø11111ØØ
&HØ2Ø4(516)	111111ØØ
&HØ2Ø5(517)	111111ØØ
&HØ2Ø6(518)	Ø1111ØØØ
&HØ2Ø7(519)	ØØØØØØØØ
&HØ2Ø8(52Ø)	ØØ11ØØØØ
&HØ2Ø9(521)	Ø1111ØØØ
&HØ2ØA(522)	11ØØ11ØØ
&HØ2ØB(523)	11ØØ11ØØ
&HØ2ØC(524)	111111ØØ
&HØ2ØD(525)	11ØØ11ØØ
&HØ2ØE(526)	11ØØ11ØØ
&HØ2ØF(527)	ØØØØØØØØ
&HØ21Ø(528)	11111ØØØ
&HØ211(529)	Ø11Ø11ØØ
&HØ212(53Ø)	Ø11Ø11ØØ
&HØ213(531)	Ø1111ØØØ
&HØ214(532)	Ø11Ø11ØØ
&HØ215(533)	Ø11Ø11ØØ
&HØ216(534)	11111ØØØ
&HØ217(535)	ØØØØØØØØ
&HØ218(536)	ØØ111ØØØ
&HØ219(537)	Ø11Ø11ØØ
&HØ21A(538)	11ØØØØØØ
&HØ21B(539)	11ØØØØØØ
&HØ21C(54Ø)	11ØØØØØØ
&HØ21D(541)	Ø11Ø11ØØ
&HØ21E(542)	ØØ111ØØØ
&HØ21F(543)	ØØØØØØØØ

↩プログラムを走らせるとまず0からzまで文字を表示し、次にその文字を0から順に横1ドットぶんずつ太くしていく

たとえばIが520のとき、行50から行70のプログラムはVRAMのデータをこんなふうに書き換えている

50 D=VPEEK(I)

D ← &BØØ1ØØØØØ

変数DにVRAMの&HØ2Ø8のデータが入る。

Ø	Ø	1	Ø	Ø	Ø	Ø	Ø

Iの現在値

&HØ2Ø8(52Ø)	ØØ1ØØØØØ
&HØ2Ø9(521)	Ø1Ø1ØØØØ
&HØ2ØA(522)	1ØØØ1ØØØ
&HØ2ØB(523)	1ØØØ1ØØØ
&HØ2ØC(524)	11111ØØØ
&HØ2ØD(525)	1ØØØ1ØØØ
&HØ2ØE(526)	1ØØØ1ØØØ
&HØ2ØF(527)	ØØØØØØØØ

60 D=D OR D¥2

D	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø
OR D¥2	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
	Ø	Ø	1	1	Ø	Ø	Ø	Ø

D OR D¥2を計算してそのデータを変数Dに入れる。

D ← &BØØ11ØØØØ

70 VPOKE I,D

D	Ø	Ø	1	1	Ø	Ø	Ø	Ø

横に1ドットぶん太くなったデータをVRAMの同じアドレスに返す。このくりかえしで文字は次々に太くなっていく。

Iの現在値

&HØ2Ø8(52Ø)	ØØ11ØØØØ
&HØ2Ø9(521)	Ø1Ø1ØØØØ
&HØ2ØA(522)	1ØØØ1ØØØ
&HØ2ØB(523)	1ØØØ1ØØØ
&HØ2ØC(524)	11111ØØØ
&HØ2ØD(525)	1ØØØ1ØØØ
&HØ2ØE(526)	1ØØØ1ØØØ
&HØ2ØF(527)	ØØØØØØØØ

キャラクタコード　文字の管理番号。たとえば、文字Ａは10進数で64、16進数で＆Ｈ４１というキャラクタコードを持っている。
【キャラクタコードの調べ方】
直接、
？　ＡＳＣ("〈文字〉")
(「？」はPRINT文とまったく同じ機能を持つ)
を実行するとその〈文字〉のキャラクタコードを答えとしてすぐ下の行に表示してくれる。また、逆に、あるキャラクタコードがなんの文字のコードかを調べるには、
？　ＣＨＲ＄(〈コード〉)
を実行すれば　その文字が表示される。
※ただし、「月」「大」や「π」、各種のケイ線など(トランプマークと黒丸、白丸は別)、グラフィックキャラクタのコードは特殊なので注意。
①キャラクタコードを調べるとき、
？　ＡＳＣ(ＭＩＤ＄("〈文字〉", ２, １)−＆Ｈ４Ø
②文字を調べるとき
？　ＣＨＲ＄(＆ＨØ１)＋ＣＨＲ＄(〈キャラクタコード〉＋＆Ｈ４Ø)
COLOR文のパラメータ　これは、
ＣＯＬＯＲ　〈前景色のカラーコード〉,〈背景色のカラーコード〉,〈周辺色のカラーコード〉
というふうに３つある。第１、第２パラメータに適当なコードを設定して実行すると、すべての文字の前景色、背景色がその値に変わる。第３パラメータの周辺色(ＳＣＲＥＥＮ１で周囲に残って見える色)とは、カラーコード０(透明)の色を決めるものだ。前景色にしろ、背景色にしろ、カラーコードが０になっていたら、COLOR文の第３パラメータで指定された色と同じ色になる。カラーコード０の色は、周辺色が見えるという意味で透明なのだ。
カラーデータ　カラーコードは０～15だから、16進数にすると＆ＨØ～＆ＨＦまで。つまり16進数１桁で表せる。そこで前景色と背景色の組み合わせは、16進数２桁で扱うことができるのだ。これも２進や16進表記のほうがわかりやすい例の１つだ。

文字パターンはもっと自由に変えられる

もちろん、もとからある文字のパターンとは関係なく、好きな文字を好きな形に変えることもできる。この場合は、リストＡのように、０と１で２進数のパターンを作って、ＶＲＡＭに書きこんでいけばいい。

38ページにも書いたが、ある文字のパターンは、ＰＧＴでは〈キャラクタコード×８〉のアドレスから８バイトぶんに入っている。リストＡの行10で変数ＡＤに設定される数値は、文字Ａのパターンがある PGT の先頭アドレスだということに注目してほしい。行10のダブルクォート(チョンチョン)で囲まれた文字を好きな文字に変えれば、その文字のパターンがある PGT の先頭アドレスがＡＤに入るようになっている。

リストＢは、同じことを、10進数とＲＥＡＤ文を使ってやったものだ。なにをやっているのかはわかりにくいが、実際にはこういう形でプログラムのなかに入っていることが多い。

```
list
100 AD=ASC("A")*8
110 VPOKE AD+0,&B01111100
120 VPOKE AD+1,&B10000010
130 VPOKE AD+2,&B00111001
140 VPOKE AD+3,&B01000101
150 VPOKE AD+4,&B01010101
160 VPOKE AD+5,&B01001001
170 VPOKE AD+6,&B01000010
180 VPOKE AD+7,&B00111100
Ok
run
Ok
color auto goto list run
```

↑リストＡをRUNさせるとこんなふうにＡがうずまき模様になる

↑左がもとの文字Ａの形。右がプログラムをRUNしたあとの形

大文字のＡをうずまき模様に変える

●リストＡ・２進数を使ったわかりやすいプログラム

```
100 AD=ASC("A")*8
110 VPOKE AD+Ø,&BØ11111ØØ
120 VPOKE AD+1,&B1ØØØØØ1Ø
130 VPOKE AD+2,&BØØ111ØØ1
140 VPOKE AD+3,&BØ1ØØØ1Ø1
150 VPOKE AD+4,&BØ1Ø1Ø1Ø1
160 VPOKE AD+5,&BØ1ØØ1ØØ1
170 VPOKE AD+6,&BØ1ØØØØ1Ø
180 VPOKE AD+7,&BØØ1111ØØ
```

●リストＢ・同じことをおこなう短い形のプログラム

```
100 FOR I=Ø TO 7
110 READ D
120 VPOKE ASC("A")*8+I,D
130 NEXT
140 DATA 124,13Ø,57,69,85,89,66,6Ø
```

こんなにかんたんに作れるカラフル文字

ＣＯＬＯＲ文を使って文字に色をつけるとすべての文字を同じ色にしかできないが、ＶＲＡＭを直接操作すれば、あらかじめ決められた８文字のグループごとに色をつけることができる。

たとえば数字の色とアルファベットの色を変えることもできるわけだ。

ＶＲＡＭの＆Ｈ２ＯＯＯから「カラーテーブル」という部分がある。ここが各８文字グループの色を記憶している部分だ。

ここでは、１バイト(16進数２桁ぶん)で２色１セットを指定する仕組みになっている。16進数の形にして見るとわかりやすいが、このデータの前半部分が「前景色」を、後半部分が「背景色」を指定しているのだ。

たとえば、カラーデータが＆Ｈ８４なら、前景色はカラーコード８(赤)になり、それ以外の部分はカラーコード４(暗い青)になる。＆ＨＦ４なら、前景色はカラーコード15(白。16進数のＦは10進数の15)、背景色はカラーコード４(暗い青)になる。

ところで、ある文字用のカラーテーブルのアドレスは、
〈その文字のキャラクタコード〉÷８＋＆Ｈ２ＯＯＯ(あまりは切り捨て)

たとえば、数字０のキャラクタコードは48、＆Ｈ２ＯＯＯは10進数の8192だから、数字０用のカラーテーブルのアドレスは、
48÷８＋8192→8198
となる。

ここを起点にして、それぞれのアドレスに16進数２桁のデータを書きこんで８文字ごとにＺまで色をつけてみたのが、右ページのプログラムだ。ただし、行10～30を加えること。

行150の数値をいろいろ変えて試してほしい。

次回のピクニックは

多色刷りの秘密に迫る！

次回は、１文字ごとに、しかも文字の１ラインごとに２色ずつ指定できる多色刷りの技法に挑戦。グラデーションのかかった文字やきれいなキャラクタデザインもできるのだ。

↑さまざまな色がつけられる多色刷り

文字につく色の仕組み

➡⬇文字のパターンの1の部分に前景色で指定された色が、0の部分に背景色で指定された色がつけられる

前景色

文字の形そのものの色のこと。フォアグラウンドカラーともいう(たんに英語でいっただけ)。COLOR文の第1パラメータでも指定できる。

⬅➡前景色と背景色をあわせると右の写真のようになる

背景色

文字の形以外の部分の色、つまり、文字どおり背景の色。バックグラウンドカラーともいう。COLOR文の第2パラメータでも指定できる。

⬆38ページのプログラムの行10〜行30(数字の0から小文字のzまでを表示する部分)と下のプログラムをあわせて実行させた結果。8文字ごとに行150のデータにあわせて色が変わっているのに注目。最後のxyzは、あとに続く大カッコなどの記号群(ここでは表示されていない)とセットで8文字グループになっているので、これだけ3文字ぶんの色つけになっている

数字の0から小文字のzに適当な色をつけるプログラム

```
100 COLOR 15,1,1
110 FOR I=0 TO 9
120 READA$
130 VPOKE 8198+I,VAL("&H"+A$)
140 NEXT
150 DATA F4,4F,84,48,8E,38,67,76,FE,8C
```

左のプログラムはかならず38ページのプログラムの行10〜行30にくっつけて使うこと。また、太文字などのプログラム全体にこのプログラムをつけると、太文字などに、指定どおりの色をつけることができる。行130は16進データを読みこむときによく使われる技法だ。

使用前(カラーテーブルの全データ)

VRAMのアドレス 16進表記(カッコ内は10進表記)	8文字ごとのカラーデータ 2進表記	カラーデータ 16進表記(カッコ内は10進表記)
&H2000(8192)	11110100	=>&HF4(244)
&H2001(8193)	11110100	=>&HF4(244)
&H2002(8194)	11110100	=>&HF4(244)
&H2003(8195)	11110100	=>&HF4(244)
&H2004(8196)	11110100	=>&HF4(244)
&H2005(8197)	11110100	=>&HF4(244)
&H2006(8198)	11110100	=>&HF4(244)
&H2007(8199)	11110100	=>&HF4(244)
&H2008(8200)	11110100	=>&HF4(244)
&H2009(8201)	11110100	=>&HF4(244)
&H200A(8202)	11110100	=>&HF4(244)
&H200B(8203)	11110100	=>&HF4(244)
&H200C(8204)	11110100	=>&HF4(244)
&H200D(8205)	11110100	=>&HF4(244)
&H200E(8206)	11110100	=>&HF4(244)
&H200F(8207)	11110100	=>&HF4(244)
&H2010(8208)	11110100	=>&HF4(244)
&H2011(8209)	11110100	=>&HF4(244)
&H2012(8210)	11110100	=>&HF4(244)
&H2013(8211)	11110100	=>&HF4(244)
&H2014(8212)	11110100	=>&HF4(244)
&H2015(8213)	11110100	=>&HF4(244)
&H2016(8214)	11110100	=>&HF4(244)
&H2017(8215)	11110100	=>&HF4(244)
&H2018(8216)	11110100	=>&HF4(244)
&H2019(8217)	11110100	=>&HF4(244)
&H201A(8218)	11110100	=>&HF4(244)
&H201B(8219)	11110100	=>&HF4(244)
&H201C(8220)	11110100	=>&HF4(244)
&H201D(8221)	11110100	=>&HF4(244)
&H201E(8222)	11110100	=>&HF4(244)
&H201F(8223)	11110100	=>&HF4(244)

前景色	背景色
F 白	4 暗い青
4 暗い青	F 白
8 赤	4 暗い青
4 暗い青	8 赤
8 赤	E 灰色
3 明るい緑	8 赤
6 暗い赤	7 水色
7 水色	6 暗い赤
F 白	E 灰色
8 赤	C 暗い緑

使用後

VRAMのアドレス	カラーデータ(2進)	カラーデータ(16進)
&H2000(8192)	11110001	=>&HF1(241)
&H2001(8193)	11110001	=>&HF1(241)
&H2002(8194)	11110001	=>&HF1(241)
&H2003(8195)	11110001	=>&HF1(241)
&H2004(8196)	11110001	=>&HF1(241)
&H2005(8197)	11110001	=>&HF1(241)
&H2006(8198)	11110100	=>&HF4(244)
&H2007(8199)	01001111	=>&H4F(79)
&H2008(8200)	10000100	=>&H84(132)
&H2009(8201)	01001000	=>&H48(72)
&H200A(8202)	10001110	=>&H8E(142)
&H200B(8203)	00111000	=>&H38(56)
&H200C(8204)	01100111	=>&H67(103)
&H200D(8205)	01110110	=>&H76(118)
&H200E(8206)	11111110	=>&HFE(254)
&H200F(8207)	10001100	=>&H8C(140)
&H2010(8208)	11110001	=>&HF1(241)
&H2011(8209)	11110001	=>&HF1(241)
&H2012(8210)	11110001	=>&HF1(241)
&H2013(8211)	11110001	=>&HF1(241)
&H2014(8212)	11110001	=>&HF1(241)
&H2015(8213)	11110001	=>&HF1(241)
&H2016(8214)	11110001	=>&HF1(241)
&H2017(8215)	11110001	=>&HF1(241)
&H2018(8216)	11110001	=>&HF1(241)
&H2019(8217)	11110001	=>&HF1(241)
&H201A(8218)	11110001	=>&HF1(241)
&H201B(8219)	11110001	=>&HF1(241)
&H201C(8220)	11110001	=>&HF1(241)
&H201D(8221)	11110001	=>&HF1(241)
&H201E(8222)	11110001	=>&HF1(241)
&H201F(8223)	11110001	=>&HF1(241)

⬆行100のCOLOR文でいったんすべての前景色を15(16進数F)、背景色を1にしているため、VPOKEしていない部分は&HF1になる

カラフルな多色刷りの原理と方法をピクピクピクニック

BASICピクニック

#2 カラフル街&H7E番地

なぜこんなにカラフルな文字ができるのか、BASICの七不思議の1つ、多色刷り。いかにも高度な技のようだがじつはかんたんにできるのだ。

BIOS(バイオス) Basic Input Output System(ベーシック・インプット・アウトプット・システム)の略。基本入出力システムと訳される。コンピュータの中核となるCPU (データを処理している部分。MSXではZ80AというLSIが使われている) と、キーボード、ジョイスティック、ディスプレイ、プリンタ、ディスクドライブなどなどの入出力装置とを結びつけるためのさまざまなマシン語プログラムの固まり。いくつかはBASICでもスイッチを切り換える感じでかんたんに利用できる。また、ほかのものもマシン語の知識があれば活用できる。

USR関数 USR関数は、DEFUSRn=〈アドレス〉という形式 (nは0から9の整数) で、0から9まで10個定義できる。nを省略すると0とみなされるが、今回の記事では、その省略された形を使っている。

パターンジェネレータテーブル 英語で書くとpattern generator table。文字パターンのデータを記憶しているVRAMの一部分。SCREEN0やSCREEN1のテキストモードでは、1文字に対し8バイトずつが割りあてられている。

カラーテーブル color table。文字の色(前景色と背景色)を記憶しているVRAMの一部分。ここを書きかえると文字の色が変わるが、SCREEN1のカラーテーブルはたった32バイトしかなく、8文字のグループごとに2色ずつしか指定できない。

多色刷りは&H7E番地で生まれる

ふつうのSCREEN1ではVPOKEを使ってもせいぜい8文字単位に2色ずつしか指定できない。それなのに、たとえば写真のようなレインボーカラーの文字を使ったプログラムにときどき出会う。文字をパターン定義して、ゲームのキャラクタや背景もそれで作っていることもある。

そういうプログラムはたいていマシン語を使っていることが多いので、多色刷りはマシン語でなければできないと思いこんでいる人もいるかもしれない。

しかし、じつは、多色刷りはすべてBASICでできるのだ。ただ、ふつうのSCREEN1ではできない。ちょっとした仕掛けが必要だ。

その仕掛けは、あらゆるMSXの本体に組みこまれているROMのなかのBIOS(バイオス)と名づけられた部分にある。

↑レインボーカラーの多色刷りの文字。31ページのプログラムを走らせたあと0からzまで表示したもの。文字ごとに色を変えることもできる

ここのアドレス&H7E(10進数で126)に、SCREEN1を多色刷りモードに切り換えるスイッチがあるのだ。そして、そのスイッチを押すためのものがBASICにはちゃんとある。

それがUSR関数という、特殊な関数だ。多色刷りモードのスイッチの入れ方だけを説明しておこう。

```
DEFUSR=&H7E
```

が、USR関数を「&H7Eのスイッチを入れる」と定義するための命令文だ。

そして、

```
A=USR(Ø)
```

として、実際に式のなかでUSR関数を使うと、その時点で、多色刷りモードに入る。

この場合、変数Aやカッコのなかの0には意味がない。USR関数が使われているということだけがたいせつなのだ。

で、このスイッチを入れるとなにが起こるか。SCREEN1のVRAMの構成が、右ページのようにダイナミックに変わってしまうのだ。

パターンジェネレータテーブル(以下PGTと略)の大きさは3倍になり、カラーテーブル(以下CTと略)の大きさはなんと、192倍にもなる。

多色刷りは、この飛躍的に大きくなったCTを利用しておこなっているのだ。

ダイナミックに変わるVRAMの構成

ふつうのSCREEN1

アドレス	内容
&H0000 〜 &H07FF	パターンジェネレータテーブル
&H1800 〜 &H1AFF	パターン名称テーブル
&H2000	カラーテーブル（&H201Fまで）
&H3FFF	

※スプライトに関係する部分などは省略しました

多色刷りのSCREEN1

アドレス	内容
〜 &H0FFF	エリア1用 / エリア2用
〜 &H17FF	エリア3用
	（パターン名称テーブル）
〜 &H27FF	エリア1用
〜 &H2FFF	エリア2用
〜 &H37FF	エリア3用

※多色刷りでは、画面を3分割し、各エリアごとに文字パターンデータとカラーデータを持っている

多色刷りの効果がわかるプログラム

```
10 SCREEN 1:WIDTH 29
20 DEFUSR=&H7E:A=USR(0)
30 FOR I=&H2000 TO &H37FF
40 VPOKE I,RND(1)*256
50 NEXT
60 LIST
```

※多色刷りモードに入り、ランダムなカラーデータをセットする

⇧ふつうのSCREEN1で、上のプログラムリストを表示

⇧しばらくすると各エリアごとにランダムな色がつく

⇧見にくいけれど上3分の1にリストの文字がちゃんと表示されている

ふつうのSCREEN1と多色刷りモードとでは、こんなにもVRAMの構成が変わる。とくに、8文字単位に1バイトぶん（前景色と背景色の指定に1バイトかかる）しか用意されていなかったCTが、3つのエリアの1文字単位に8バイトぶん（文字のパターン1ラインごとに1バイト）が使えるように拡張されている点に注目してほしい。

パターンジェネレータテーブル

パターンジェネレータテーブル

⇧文字のパターンのセットが入っている部分　⇨多色刷りでは画面3分の1ごとに各エリア用のセットがある。ただし、ここでは後半3分の2にはなにも入っていない

パターン名称テーブル

パターン名称テーブル

カラーテーブル

カラーテーブル

⇧⇗（1つ上の写真）パターン名称テーブルは画面と1対1対応しているので、変化はない。写真は右上のプログラムのリストを表示させた場合の状態を文字の形で表現したもの　⇧⇨CTはふつう32バイトしかないが、多色刷りでは6K（6144）バイトぶん、192倍に拡張される。各文字のパターン1ラインごとに2色（前景色、背景色）を指定できる仕組みになっているのはこのためだ

VDP　Video Display Proccessor(ビデオ・ディスプレイ・プロセッサ)の略で画面表示用LSIのこと。とくに、MSX2のVDPに使われているLSIには、MSX-VIDEOという愛称がある。MSXの画面表示はこのVDPに牛耳(ぎゅうじ)られているのだが、めったに表に出てこないので、存在に気がついていない人も多い。いわば、VRAMの頭脳のようなものだ。ところで、VDPにちょっかいを出すためのVDP(n)というシステム変数があり、これがまたおもしろいのだが、それはまた別の機会に。

パターン名称テーブル　パターンネームテーブル(pattern name table)ということもある。画面と1対1対応していて、画面のある部分に文字があると、パターン名称テーブルの対応する部分にはその文字のキャラクタコードが入っている。逆に、パターン名称テーブルのあるアドレスに数値を書きこむと、その数値をキャラクタコードとする文字が画面の対応する部分に表示される。ところで、LOCATE文で文字の置ける範囲は限られているが、パターン名称テーブルを使えばたとえばファンクション表示の上にも文字を置くことができる。

キャラクタコード　文字に割りふられた整理用番号。通常、キャラクタコード0から31はコントロールコードという特殊な制御コードに割りあてられているが、VRAMではキャラタコード0から31は「月」などのグラフィックキャラクタに割りあてられている。VRAMでのキャラクタコードのことをパターン番号ということもある。このへんのことは、67ページの「ファンダムハウス」でちょっと悩んでいる。

MSX1ユーザーへ　この記事の写真にはグラフィックキャラクタの一部分(「木」以降4文字)に妙な色のついているものがある。これはMSX2/2+の特性で、MSX1ではふつうの文字でしか表示されない。しかし、いったん多色刷り用のカラー定義をすれば、MSX1もMSX2／2+もまったく変わりがないのでご心配なく。

MSX2/2+ユーザーへ　右ページにあるカラフル斜体文字プログラムに次の行15を加えてみよう。

```
15 SCREEN 0:WIDTH 80:WIDTH 40
```

プログラムが走りはじめたときに画面の上3分の1だけでなく、画面ぜんぶの文字が見えるようになる。これは、エリア2、3用のPGTと、SCREEN0の80字モード、40字モードのPGTがたまたま一致していて、モードを切り換えるたびに文字のパターンがVRAMに残っていくからだ。

SCREEN1のふりをしたSCREEN2

ところで、多色刷りSCREEN1のVRAMは、SCREEN2のそれとそっくりなのだ。それも当然。多色刷りモードのスイッチを入れるということは、BIOSの「VDPのみをグラフィック2(SCREEN2)モードにする」という部分を実行することだからだ。かんたんにいうと、「画面表示の仕組みだけをSCREEN2と同じにする」ということになる。

ふつう直接SCREEN2という命令文を実行すると、とたんにテキストモードにもどってくるが、これはBASICの性格のせいだ。しかし、VDPだけをSCREEN2にした場合は、BASICにとってあくまでここはSCREEN1というテキストモードのままだ。つまり、BASICに対してSCREEN1のふりをし、VDPに対してはSCREEN2としてふるまっていることになる。

SCREEN2の画面表示の仕組みはなかなかおもしろい。

SCREEN2にもいちおうパターン名称テーブル(以下PNTと略)があるが、じつはここには0～255の数値がこの順で3回くりかえして入れられたまま変化しない(文字1セットぶんは画面のちょうど3分の1ずつに置かれる)。つねに、キャラクタコード0から255の文字が3セット、固定して表示されているだけなのだ。

この文字のセットには、それぞれ独立したPGTが対応していて、画面3分の1ずつが別々のPGTを持っている。

↑すべての文字をカラフルな斜体文字にするプログラムを実行してリストを表示させてみた。すべてBASICでここまでできるのだ

SCREEN2を実行すると、PGTをすべて0にしてしまうので、最初はなにも表示されていないように見える。そこでたとえば円をかくとどうなるか。ここがおもしろい。その円の線が通過する部分の各文字を、全体として円になるようにパターン定義していくのだ。同時にCTも指定どおりに変えていく。

下のカコミのプログラムは、その仕組みを実験したものだ。SCREEN2で円を書き、そのまま、SCREEN1にもどって多色刷りモードに入ってから、キャラクタコード0から255の文字をPNTに直接3セットぶん書きこんでいる。つまり、SCREEN2でのPNTを再現しているわけだ。すると、SCREEN1に円が現れる！

画面の上3分の1が変なのは、SCREEN1を実行したとき、自動的にPGTのエリア1に相当する部分に文字パターンが書きこまれるからだ。CTはまえのままなので、色だけが円の部分を構成している。

逆にいうと、多色刷りモードに入ったときは、エリア1のぶんのPGTしか文字のパターンを持っていないことになる。だから、エリア2、3用のPGTにも文字パターンを書きこんでおかないと文字が見えないのだ。

この作業をBASICでやると時間がかかるので、その部分だけをマシン語でやっているプログラムが多い。多色刷りをマシン語でやっているように見えたのはそのせいだったのだ。

SCREEN1とSCREEN2の密会現場

SCREEN1で円を見るプログラム

```
10 COLOR 15,4,7:SCREEN 2
20 CIRCLE (127,95),95
30 SCREEN 1:KEYOFF
40 DEFUSR=&H7E:A=USR(0)
50 FOR I=&H1800 TO &H1BFF
60 VPOKE I,I MOD 256
70 NEXT
80 GOTO 80
```

↑行20を実行しおえたところ。SCREEN2で円をかいた

↑行70を実行しおえたところ。SCREEN1にも円が現れる

BASIC テクニック

すべての文字をカラフルな斜体文字にするプログラム

初期設定

行10は、このプログラムを実行するまえにSCREEN2を実行していれば、なくてもかまわない。そうでない場合は、最初しばらくのあいだまったく文字が見えなかったり、8文字だけが妙な色で見えていたりする。行30がこの記事の全プログラムに共通して使われている多色刷りモードに入る部分。

画面を文字で埋める

ここは、たんにキャラクタコード0から255までの文字を画面いっぱいに表示しているだけ。30ページのプログラムの行50〜70とまったく同じだ。&H1800というアドレスは、SCREEN1のPNT(パターン名称テーブル)の先頭アドレスで、ここから1バイトずつ、0〜255の数値を順に書きこんでいる。PNTの大きさは256×3バイトなので、MODを利用して3セット書きこんでいる。画面には最初、上3分の1でしか文字が見えないが、エリア2、3用のPGTに文字パターンが定義されていないためだ。MSX2以上ならぜんぶ見えるようにする方法がある(左ページの注記参照)ので試してみよう。

↑上3分の1しか文字が見えないが、実際には下にも文字は置かれている

色をつけていく

&H2000はCT(カラーテーブル)の先頭アドレス。ここから1バイトずつカラーデータを書きこんで色をつけている。1文字は横8ドット×縦8ラインで構成されているが、多色刷りのCTは、1ラインごとの前景色と背景色の組み合わせを1バイトとして、1文字につき8バイトずつ割りあてられている。ただし、ここもエリア1〜3用に分割されているので注意。

↑キャラクタコード順に次々と色がついていく。エリア2、3の文字はぜんぶ背景色(黒)になっている

斜体文字にする／エリア1〜3にパターン書きこみ

&H7FFは、PGT(パターンジェネレータテーブル)のうち、エリア1用のPGTの最終アドレス。エリア1の文字パターンを行110、120で斜体処理しながら(この部分は先月号で紹介したプログラムと同じ)、その結果を行130でエリア1〜3に書きこんでいる。画面では、エリア1が斜体文字になっていくのと同時にエリア2、3にも同じ文字が表示されていくように見えるはずだ。

↑斜体文字の処理をしてエリア2、3にも同じパターンを書きこんでいく。ここではじめて前景色の色が出てくる

```
10 COLOR 15,1,15:SCREEN 2
20 SCREEN 1:WIDTH 32:KEYOFF
30 DEFUSR=&H7E:A=USR(Ø)
40 FOR I=Ø TO 256*3-1
50 VPOKE I+&H18ØØ,I MOD 256
60 NEXT
70 FOR I=&H2ØØØ TO &H37FF
80 VPOKE I,(I MOD 8 +5)*16+1
90 NEXT
100 FOR I=Ø TO &H7FF
110 D=VPEEK(I)
120 D=D OR D¥2:D=D¥VAL(MID$("84442211",I
MOD 8 +1,1))
130 FOR J=Ø TO 2:VPOKE I+&H8ØØ*J,D:NEXT
140 NEXT
```

エリア1にいた「A」の場合

エリア1の文字Aに注目すると、まず1ラインごとに色をつけられ、次に斜体文字に変化していく。

イラスト文字のサンプルプログラム「虹と白い雲」

苦労をおしまなければイラストふうの文字も作れる。行1000〜1070は文字ac用のデータ、行1080〜1150は文字bd用のデータ。0と1だけのデータは、8桁ずつが左の文字のパターン、右の文字のパターンになっていて、あとに続く2つの数値(16進数)でそのラインの前景色と背景色を1文字の1ラインずつ指定している。

```
10 COLOR 15,1,15:SCREEN 2:SCREEN 1
20 DEFUSR=&H7E:A=USR(Ø)
30 A$="ac"+CHR$(29)+CHR$(29)+CHR$(31)+"b
d":AD=ASC("a")*8
40 LOCATE 1Ø,1:PRINT A$
50 RESTORE 1ØØØ
60 FOR I=Ø TO 15
70 READ P$,C1$,C2$
80 P1=VAL("&B"+LEFT$(P$,8)):P2=VAL("&B"+
RIGHT$(P$,8)):C1=VAL("&H"+C1$):C2=VAL("&
H"+C2$)
90 VPOKE AD+I,P1:VPOKE AD+I+&H2ØØØ,C1
100 VPOKE AD+I+16,P2:VPOKE AD+I+&H2ØØØ+1
6,C2
110 NEXT
1000 DATA 1111ØØØØØØØØØØØØ,86,64
1010 DATA ØØØØ11111111ØØØØ,89,64
1020 DATA 1111ØØØØØØØØ1111,A9,68
1030 DATA ØØØØ11111111ØØØØ,A2,98
1040 DATA 1111ØØØØØØØØ1111,72,9A
1050 DATA ØØØØ11111111ØØØØ,75,2A
1060 DATA 1111ØØØØØØØØ1111,D5,27
1070 DATA ØØØØ11111111ØØØØ,D4,57
1080 DATA ØØØØØØØØØØØØ1111,F4,5D
1090 DATA ØØØØØØØØØØØØ1111,F4,D4
1100 DATA ØØØØØØØØØØØØØØØØ,F4,F4
1110 DATA ØØØØ11ØØØØØØØØØØ,F4,F4
1120 DATA 1ØØ1111ØØØØØØØ11,F4,F4
1130 DATA 11111111ØØØØØ111,F4,F4
1140 DATA 11111111ØØØØØØ11,F4,F4
1150 DATA 1111111ØØØØØØØØØ,F4,F4
```

↑左のプログラムを実行した結果。ほんの4文字ぶんのCG

文字センサからアニメーションまで遊べるパターン名称テーブル

BASICテクニック

＃3 画面に映ったVRAM

パターンジェネレータテーブルやカラーテーブルに比べると効果が目に見えにくいので、なんとなく日陰者のようなパターン名称テーブルのお話。

VRAMのなかにあるもうひとつの「画面」

まえの2回では、ほとんど触れることのなかったパターン名称テーブル(Pattern Name Table：以下PNTと略)について蘊蓄(むずかしい字だ)を傾けてみよう。

PNTは、SCREEN1の場合、ふつう&H1800からはじまる768バイトのエリアになる。ところで、SCREEN1での1画面は32字×24行＝768字なのだが、この数字が一致しているのは偶然ではない。

画面を1文字ぶんのマスで区切ったとすると、PNTの1バイトは、そのマスの1つ1つと対応しているのだ。右ページのように、ちょうど画面がそっくりVRAMのなかに入った感じになっている。というより、このPNTにキャラクタコードが書きこまれることによって、画面に文字が表示されているのだ。ただし、スプライトはまったく別のところで管理されている。

PNTのてきとうなアドレスにVPOKEで数値(キャラクタコード)を書きこめば画面に文字が表示できるし、てきとうなアドレスの内容をVPEEKで調べれば、その位置になんの文字があるかを判断することができる。とくに、ファンダムに掲載しているプログラムなどでは後者の使い方が圧倒的に多い。

左のプログラムは、画面の好きな位置をスプライトの箱で指せば、そこにある文字のキャラクタコードを表示するものだ。行10から30がスプライトパターン定義などの初期設定、行40から行70がカーソルキーの入力をスプライト座標に変換して表示しているところ、そして行80がこのプログラムの核だ。右ページの変換公式から計算したアドレスにある数値を、16進数にして、行90で表示しているだけだが、その数値こそ、そこにある文字のキャラクタコードなのだ。画面のあちこちに箱を動かして遊んでみよう。

↑文字センサの実行画面。好きな場所にカーソルを持っていこう

キャラクタコード　文字の管理用番号。グラフィックキャラクタ以外の文字は、
PRINT　CHR$(〈コード〉)↵
VPOKE　&H～,〈コード〉↵
※ただし、&H～にはパターン名称テーブル内のアドレスを入れる
のどちらでも同じ文字が表示される。しかし、グラフィックキャラクタはちょっと違う。VPOKE文ではVRAM用キャラクタコード表を使えば上の形でいいが、PRINT文では、
PRINT　CHR$(1)+CHR$(〈コード〉+&H4Ø)↵
としないと表示されない。

文字センサプログラム　行5のREM文はこのプログラムを使うための初期設定用だ。行5だけをいったん表示して、行番号と「'」をスペースキーで消してからリターンキーを押すと画面が初期化されるので、その後リストを表示するなどしてからRUNすること。

文字センサ(キャラクタコード検出)プログラム

```
5 'SCREEN 1,Ø:WIDTH 29
10 KEYON:LOCATE Ø,22
20 PRINT "そのモシ゛のキャラクタコート゛は、&H  て゛す。";
30 SPRITE$(Ø)=STRING$(7,129)+CHR$(255)
40 S=STICK(Ø)
50 X=X+(S=3)*(X<28)-(S=7)*(X>Ø)
60 Y=Y+(S=5)*(Y<23)-(S=1)*(Y>Ø)
70 PUTSPRITE Ø,(X*8+15,Y*8-1),8
80 A$=HEX$(VPEEK(&H1800+X+Y*32+2))
90 LOCATE 19,22:PRINT A$;
100 GOTO 40
```


※この記事のプログラムはすべてのMSX MSX2/2+で動きます。

パターン名称テーブルと画面の関係

下にずらっとならんだ数字は、初期画面(右の写真)でのPNT(パターン名称テーブル)の状態を表している。2桁の数字は、PNTの1バイト(アドレスの単位)に入っているデータを16進数で表記したもので、左上すみが&H1800番地、その1つ右が&H1801番地……、2行目の左端が&H1820番地というふうに32バイトごとにならべてある。

目を薄めて見れば、写真の画面と数字のならび方の状況がそっくりであることがわかるだろう。この2桁の16進数は、それぞれの場所にある文字のキャラクタコードになっているのだ。いちばん多い「20」にわざと青い色をつけておいたが、これはキャラクタコード&H20の文字、すなわち空白だ。いちばん下の行にはところどころに「20」以外の数字があるが、ここはファンクションキーの内容を表示しているところなのだ。

暇な人は1つ1つのデータを下のキャラクタコード表で調べて確認してみよう。

画面

```
MSX BASIC version 2.0
Copyright 1985 by Microsoft
23433 Bytes free
Disk BASIC version 1.0
Ok
█



color auto  goto  list  run
```

VRAMパターン名称テーブル

アドレス	1つのアドレス単位のメモリの内容																														
&H1800	20	20	4D	53	58	20	42	41	53	49	43	20	76	65	72	73	69	6F	6E	20	32	2E	30	20	20	20	20	20	20	20	20	20
&H1820	20	20	43	6F	70	79	72	69	67	68	74	20	31	39	38	35	20	62	79	20	4D	69	63	72	6F	73	6F	66	74	20	20	20
&H1840	20	20	32	33	34	33	33	20	42	79	74	65	73	20	66	72	65	65	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1860	20	20	44	69	73	6B	20	42	41	53	49	43	20	76	65	72	73	69	6F	6E	20	31	2E	30	20	20	20	20	20	20	20	20
&H1880	20	20	4F	6B	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H18A0	20	20	FF	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H18C0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H18E0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1900	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1920	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1940	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1960	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1980	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H19A0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H19C0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H19E0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1A00	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1A20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1A40	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1A60	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1A80	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1AA0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1AC0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
&H1AE0	20	20	63	6F	6C	6F	72	20	61	75	74	6F	20	20	67	6F	74	6F	20	20	6C	69	73	74	20	20	72	75	6E	20	20	20

SCREEN1での座標とアドレスの変換公式

座標　X,Y

アドレス　&H1800+X+Y*32+?

画面の場所とPNTのアドレスとの関係は上の図を見ればわかるが、LOCATE文の座標との関係はちょっとしたくふうがいる。というのも、LOCATE文はWIDTHの数値によって同じ座標でも指し示す場所が変わるからだ。たとえば、座標(0,0)は、WIDTH32だと左上すみだが、WIDTH29(初期状態)では左上すみから2文字ぶん右側になる。

落ち着いて考えればあたりまえだが、WIDTH32のとき(つまり画面いっぱいに使っているとき)は、座標(X,Y)とPNTのアドレス「&H180Ø+X+Y×32」が同じ場所になるが、WIDTH29の場合は、このアドレスに+2しなければいけない(最初の2文字ぶんのずれを修正する)。

上の変換公式の「?」はこのずれの修正用だ。WIDTH32では0、30〜31では1、28〜29では2……というふうにWIDTHが2字ぶんちぢまるごとに加えるべき「?」は1ずつ増えていく。VPEEKでテキスト座標を使うときはこの公式を使って変換しよう。

VRAM用キャラクタコード表

16進数上1桁 →／16進数下1桁 ↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		π		0	@	P	`	p	♠	9	A	ー	タ	ミ	た	み
1	月	┴	!	1	A	Q	a	q	♥	あ	。	ア	チ	ム	ち	む
2	火	┬	"	2	B	R	b	r	♣	い	「	イ	ツ	メ	つ	め
3	水	┤	#	3	C	S	c	s	♦	う	」	ウ	テ	モ	て	も
4	木	├	$	4	D	T	d	t	○	え	、	エ	ト	ヤ	と	や
5	金	┼	%	5	E	U	e	u	●	お	・	オ	ナ	ユ	な	ゆ
6	土	│	&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ
7	日	─	'	7	G	W	g	w	ぁ	き	ァ	キ	ヌ	ラ	ぬ	ら
8	年	┌	(	8	H	X	h	x	ぃ	く	ィ	ク	ネ	リ	ね	り
9	円	┐	)	9	I	Y	i	y	ぅ	け	ゥ	ケ	ノ	ル	の	る
A	時	└	*	:	J	Z	j	z	ぇ	こ	ェ	コ	ハ	レ	は	れ
B	分	┘	+	;	K	[	k	{	ぉ	さ	ォ	サ	ヒ	ロ	ひ	ろ
C	秒	╳	,	<	L	¥	l	¦	ゃ	し	ャ	シ	フ	ワ	ふ	わ
D	百	大	-	=	M	]	m	}	ゅ	す	ュ	ス	ヘ	ン	へ	ん
E	千	中	.	>	N	^	n	~	ょ	せ	ョ	セ	ホ	゛	ほ	※
F	万	小	/	?	O	_	o		っ	そ	ッ	ソ	マ	゜	ま	■

表の見方 コードを調べたい文字が表のどの位置にあるかをまず調べる。横方向の数字(16進数上1桁)と縦方向の数字(16進数下1桁)をこの順にならべて、頭に「&H」をつければ、それがその文字のキャラクタコードだ。たとえば、「A」は横方向「4」と縦方向「1」のマスにあるので、キャラクタコードは&H41となる。

※表中、赤い文字は識別用に入れたもので、実際の画面では空白になります。

※この記事では、すべてSCREEN1の場合について説明しています。ほかのスクリーンモードでは、この記事で説明している方法、およびプログラムは適用されません

たくさんの「画面」でアニメーション

いろいろな機能のテーブル　まえの2つについては5月号、6月号のくりかえしになるが、パターンジェネレータテーブルは文字のパターンのデータが入っている部分、カラーテーブルは文字のカラーデータが入っている部分、スプライト属性テーブルは、スプライトの座標、パターン番号、色が入っている部分、スプライトジェネレータテーブルはスプライトパターンのデータが入っている部分だ。
マリック　マジックとトリックの合成語。それにしても、ミスター・マリックはいつ見てもすごい。ここでやっていることは、マリックさんの足もとにもおよばないが、まあいいじゃないか。
見えているPNT　ディスプレイページとか、ビジュアルページということもある。ちなみに「見えているPNT」ということばは、この記事でかってに作ったことば。
PRINTできるPNT　アクティブページということもある。「PRINTできるPNT」ということばもかってに作った。ごめん。ちなみに、MSX 2以上では、ディスプレイページとアクティブページを、SETPAGEという命令でかんたんに切り換えられるようになっている。ただし、これはSCREEN 5以降での話。
システム変数　MSXのシステムに組みこまれている特殊な変数。VDP (n) というシステム変数は一般に画面表示のなにがしかのシステムと関わりがある。ほかに、システム変数には、有名なSPRITE$ (n) (スプライトパターン定義用)、BASE (n) (VRAMの各テーブルの先頭アドレス用)、TIME(タイムカウント用)、ERR(発生したエラーのコード)、ERL(エラーの発生した行) などがある。システム変数のなかには、代入できないものもあったり、へたな数字を代入するとわけがわからなくなったりするのがあり、使うときは慎重に。たとえば、BASE(5)というシステム変数は、VDP (2) と似ていて、アドレスを代入すれば「見えているPNT」を別のエリアに移す機能があるが、VDP (2) と違って、いったん代入してしまうと、SCREEN文を実行しても画面の状態はもとにもどってくれない。もう一度、もともと持っていた値を代入すればいいのだが、そうしようと思っても打ちこんだ文字は画面に表示されないのでなかなかむずかしい。ただ、これも電源を切ればもとにもどるので、興味のある人は試してみよう。そのかわり、代入することのできるアドレスは、右ページの図の各エリアの先頭アドレスのみ。これ以外の数値を代入しようとするとエラーになる。
POKE文　VRAM関係での話なので VPOKEとまちがう人もいるかもしれないが、Vはないので注意。この記事の場合は、マシン語とは関係ないが、この命令文はアドレスや数値などをへたにまちがえると、暴走(MSXが気絶すること) することがあるので、慎重に扱おう。
?　PRINT文の省略形。ダイレクトモードで変数の内容を調べるときには、いちいち「PRINT」と打ちこむのがめんどうなので、この「?」をよく使う。ほんとによく使う。

■SCREEN 1のあきエリア

SCREEN 0～3用のVRAM(つまりMSX 1～MSX 2+まで共通にあるVRAM)の容量は16Kだ。小さいようだが、SCREEN 1のVRAMマップ(マニュアル、または5～6月号のBASICピクニック参照)を見ると、PNT(パターン名称テーブル)のほかにパターンジェネレータテーブルやカラーテーブル、スプライト属性テーブル、スプライトジェネレータテーブルなどいろいろな機能のテーブルが入っているのに、かなりの部分がスカスカにあいているのだ。

SCREEN 2では、けっこう有効に活用しているのだが、SCREEN 1(SCREEN 0や3も同様)では、この部分をまったく使っていない。

しかし、ちょっとしたマリックを使えば、SCREEN 1でもこのあきエリアを活用することができる。たとえば、このあきエリアとPNTのあまり知られていない性質を利用すれば、かんたんなアニメーションプログラムが作れるのだ。文字をならべた画面をパタパタアニメの原理で切り換えるだけなのだが、これがけっこうおもしろい。ここでは、そのやり方と原理をかんたんに説明しよう。

■見えているPNT

やり方を説明するまえに、まず最初に、

KEY 1,"SCREEN 1"+CHR$(13)⏎

を実行しておこう。これから説明することを試していると、ちょっとした数値の打ち間違いですぐにわけのわからない画面になる。そんなときはF1キーを押せばもとにもどるわけだ。

さて、じつは、PNTには「見えているPNT」と「PRINTできるPNT」の2つがある。ふつうに使っているPNTは、この両方が一致しているためにとくになんとも感じないが、これを別々にてきとうなアドレスに移せるのだ。

見えているPNTを指定するのが、VDP(2)というシステム変数だ。これは、PNTのいちばん最初のアドレスをいわば番号で指定しているもので、初期状態では、VDP(2)=6になっている。6番とは、&H1800からはじまるエリアのことだ。このVDP(2)に別の値を代入すれば、見えているPNTが別のアドレスに移動する。SCREEN 1では、0番から15番までになる、ほかの機能に使われているエリアを除くと、右ページの図のように、11個のエリアが使える。

■PRINTできるPNT

PRINTできるPNTのアドレスを変えるにはPOKE文を使う。

POKE &HF923, ～

の～の部分に、持っていきたいエリアの最初のアドレス(16進数)の上2桁を指定すればいい。複雑そうだが右の図を見ればかんたんだ。たとえば、最初のPNTの位置は、

POKE &HF923,&H18⏎

を実行したのと同じ状態になっている。「&H18」は、このPNTの最初のアドレス「&H1800」の最初の2桁なのだ。

この2つのPNTの切り換えを利用したアニメーションプログラムが下のリストだ。打ちこんで走らせてみると、サインカーブがなめらかに波打つように見えるはずだ。

これは11個のエリアに連続パターン(右ページの11枚の画面写真)をかき、見えているPNTを連続して切り換えて動いているように見せているのだ。

このプログラムで、文字を書きこんでいる部分(行80のFOR～NEXTループ)を別のプログラムに置き換えれば、オリジナルアニメーションがいくらでもできる。ぜひ、チャレンジしてほしい。

プログラム解説

【おもな変数の意味】●S (n)→サインカーブをかくためのY座標増分●R (n)→VDP(2)で指定するためのあきエリア番号
【おもな部分の解説】●10～30→初期設定●40～50→各配列変数の設定●60, 90→行70～80をくりかえすループ●70→2つのPNTを両方とも配列変数R (n)の内容に従って切り換え、そのときのPNTをまずCLS●80→2文字ぶんずつずらしたサインカーブを各PNTにかく●100～120→見えているPNTを順に切り換え、永遠にくりかえす●130→スクリーンの状態をもどして終了(CTRL+STOPを押すとこの行を実行)

サインカーブのアニメーション

```
10 SCREEN 1:COLOR 15,4,7:KEYOFF
20 ON STOP GOSUB 130:STOP ON
30 DEFINT A-Z:DIM S(21),R(10)
40 FOR I=0 TO 21:READ S(I):NEXT
50 FOR I=0 TO 10:READ R(I):NEXT
60 FOR C=0 TO 10
70 VDP(2)=R(C):POKE &HF923,R(C)*4:CLS
80 FOR I=C*2 TO C*2+21:LOCATE 3+I-C*2,11
+S(I MOD 22):PRINT "●":NEXT
90 NEXT
100 FOR C=0 TO 10
110 VDP(2)=R(C):FORW=0TO100:NEXT
120 NEXT:GOTO100
130 SCREEN 1:END
140 DATA 0,3,5,8,9,10,10,9,8,5,3,0,-2,-4
,-7,-8,-9,-9,-8,-7,-4,-2
150 DATA 2,3,4,5,6,7,9,10,11,12,13
```

11枚の画面を作るPNTたち

SCREEN1のVRAMマップ

&H0000
&H0800
&H0C00
&H1000
&H1400
&H1800
&H1C00
&H2000
&H2400
&H2800
&H2C00
&H3000
&H3400
&H3800

PNTを切り換える2つのスイッチ

POKE&HF923, &H08
VDP(2)=2
POKE&HF923, &H0C
VDP(2)=3
POKE&HF923, &H10
VDP(2)=4
POKE&HF923, &H14
VDP(2)=5
POKE&HF923, &H18
VDP(2)=6
POKE&HF923, &H1C
VDP(2)=7
POKE&HF923, &H24
VDP(2)=9
POKE&HF923, &H28
VDP(2)=10
POKE&HF923, &H2C
VDP(2)=11
POKE&HF923, &H30
VDP(2)=12
POKE&HF923, &H34
VDP(2)=13

VDP(2)=~

SCREEN1の初期状態では、
VDP(2)=6
になっている。中身を見るために、
?VDP(2)⏎
を実行すると「6」が表示される。
VDP(2)のなかに入っている数値は、いったいなにものなのかというと、SCREEN1のVRAMを&H400ごとに区切って0から15まで番号をつけたものだと考えていい。そして、現在「見えるPNT」になっているエリアの番号が、VDP(2)のなかに入っているのだ。
このVDP(2)に自分の使いたいエリアの番号を代入すると、そのエリアに入っているデータが画面に表示される。つまり、指定されたエリアが「見えているPNT」になるわけだ。ためしに、
VDP(2)=0⏎
などとしてみると、カーソルが消え、奇妙な文字がたくさん現れる。これは、パターンジェネレータテーブルのデータが文字に置き換えられて表示されているのだ。
左の本文で設定しておいたF1キーを押せばもとにもどる。

POKE&HF923,~

POKE &HF923, ~
は「PRINTできるPNT」の位置を変える命令だ。「~」の部分には移動したい先のアドレス(16進数4桁)の上2桁を書きこめばいい。また、そのときのVDP(2)の値の4倍を書きこめば、2つのPNTが一致するという法則がある(リストの行70参照)。
左の図で、それぞれのエリアから引き出された線の上と下に書いてあるのが、そのエリアをPNTとして使うための命令だ。上下の2つともを同時に(コロンでつなぐ)実行すれば、ごくふつうにリストを見たり、プログラムを打ちこんだりできるのだ。
あきエリアでない0、1、8を指定すると奇妙なことが起こる。どう間違えても機械は壊れないので試してみよう。POKE文で打ちミスをすると暴走することもあるけれど、どうしようもなくなったら電源を切ればいいのだ。
ただし、いろんな実験をするまえに、打ちこんだプログラムだけはセーブしておくのが常識だ。

どういうわけかBASICはスプライトにやさしい

BASICテクニック

＃4　VRAMのなかの妖精

＃1から＃3までずっと文字とVRAMの関係を追ってきたが、今回はもともと「妖精」という意味のあるスプライトに視点を移してみよう。

SPRITE$とPUTSPRITE

文字でキャラクタを作るには、VRAMに直接データを書きこむしか方法がなかったが、スプライトは違う。スプライトは、そのパターンを定義したり、色を指定したりするための専用のことばがBASICのなかに用意されているのだ。それが、パターンを作るSPRITE$(n)と、スプライトを表示するPUTSPRITE命令だ。

下図のように、SPRITE$(n)にCHR$関数を使って1ラインごとのパターンデータ(8桁の2進数で形を表す)を入れると、パターン番号nのスプライトパターンが定義できる。

パターンデータはVRAMのスプライトパターンジェネレータテーブル(&H3800からの2048バイト)に、パターン番号の順に書きこまれる。この仕組みは文字のパターンジェネレータテーブルとそっくりで、じっさいスプライトパターン番号はスプライト版キャラクタコードだと考えていい。

PUTSPRITE命令で面番号、座標、色、パターン番号をセットで指定するとスプライト属性テーブル(&H1B00から始まる128バイト)の内容が書き換えられ、スプライトが表示される。このテーブルは4バイト1セットで1つのスプライトを表していて、スプライト面番号とは、このテーブルの何セット目のデータに支配されるスプライトかを表しているのだ。

SPRITE$(n)　いっけん文字関数のように見えるが、じつはシステム変数。これに文字列を代入することでパターンを定義する。

?　SPRITE$(Ø)⏎

とすれば内容の文字列が表示される(ただし、コントロールコードがまじっているといろいろ変なことが起こる)。また、ふつうの文字列のようにこれに対して文字関数も使える。

スプライトパターンジェネレータテーブル　sprite pattern generator table。8×8ドットのスプライトなら、パターン番号順に8バイトずつのデータがならび、ぜんぶで256個作れる(文字と同様)。16×16ドットなら32バイトごとにならび、ぜんぶで64個作れる。

スプライト属性テーブル　sprite attribute table。PUTSPRITE命令で指定した数値はここに収められ、スプライトが表示される。

●スプライトパターン定義の基本的な書式　※8×8ドットのパターンの場合

```
SPRITE$(p)=CHR$(d1)+CHR$(d2)+....+CHR$(d8)
```

p：スプライトパターン番号
d1〜d8：スプライトパターンデータ

●スプライト表示の基本的な書式

```
PUTSPRITE m,(x,y),c,p
```

m：スプライト面番号
(x,y)：スプライト座標
c：カラーコード
p：スプライトパターン番号

この2つの指定は省略可。省略すると初期状態ではcが15(白)、pが面番号の数値になる。まえに指定していればそれが生きる。

※この記事のプログラムはすべてのMSX MSX2/2+で動きます。

おなじことをする3つのプログラム

タイプ1：2進データ方式

```
10 SCREEN 1,1
20 FOR I=0 TO 7
30 READ A$
40 SP$=SP$+CHR$(VAL("&B"+A$))
50 NEXT
60 SPRITE$(0)=SP$
70 PUTSPRITE 0,(120,100),10
100 DATA 00111100
110 DATA 01111110
120 DATA 11111111
130 DATA 11011011
140 DATA 11011011
150 DATA 11011011
160 DATA 01111110
170 DATA 00000000
```

タイプ2：10進データ方式

```
10 SCREEN 1,1
20 FOR I=0 TO 7
30 READ A
40 SP$=SP$+CHR$(A)
50 NEXT
60 SPRITE$(0)=SP$
70 PUTSPRITE 0,(120,100),10
100 DATA 60,126,255,219,219,219,126,0
```

タイプ3：文字列方式

```
10 SCREEN 1,1
20 SPRITE$(0)="<~■□□□~"
30 PUTSPRITE 0,(120,100),10
```

※タイプ3行20の「■」は、KEY1,CHR$(255)を実行し、F1キーで打ちこみます

Ok

⬅3つのタイプのどれでもこの画面になる

どのプログラムでもおなじ画面になる

スプライトパターン定義のサンプルプログラムを3タイプ作ってみた。3つともおなじ結果になる。

緑の線で囲まれた部分に注目してほしい。タイプ1では、パターンがよくわかる2進数のデータになっているが、タイプ2ではそれを10進数にしてデータ部分を短くし、タイプ3ではさらにCHR$を文字そのものに置き換えて直接SPRITE$に代入し、圧倒的に短いプログラムにしている。

タイプ3のやり方はよく使われるものだが、ひとつだけ大きな欠点がある。それは、0～31と127のデータがあると使えないということだ。この数値はコントロールコードなので、ふつうのやり方では文字の形に置き換えることができないからだ。たとえば、31ページでやっているスプライトのパターンデータのなかには、この範囲の数値がたくさんあるためにタイプ3の方法は使えない。

スプライト属性テーブルの変化

アドレス	内容〈2進表記〉	=〈16進表記〉(〈10進表記〉)	
&H1B00	11010001	=&HD1(209)	Y座標
&H1B01	00000000	=&H00(0)	X座標
&H1B02	00000000	=&H00(0)	パターン番号
&H1B03	00001111	=&H0F(15)	カラーコード

⬇

PUTSPRITE 0,(120,100),10

⬇

アドレス	内容〈2進表記〉	=〈16進表記〉(〈10進表記〉)	
&H1B00	01100100	=&H64(100)	
&H1B01	01111000	=&H78(120)	
&H1B02	00000000	=&H00(0)	※1
&H1B03	00001010	=&H0A(10)	※2

※1　例のPUTSPRITE命令のようにパターン番号の指定を省略すると、まえの状態(ここでは初期状態：面番号とおなじ数値)のままになる。

※2　この部分は下位4ビットだけを使ってカラーコードを表す。上位4ビットのうち、最上位のビットはX座標がマイナスのときに用いるが、上位のほかの3ビットはつねに0。

※3　データの0は文字に置き換えられないが、ここでは「最後の文字を指定しない」という方法で結果的に0をセットしたことになる。本来なら文字列は標準サイズの場合、8文字でなければならないのに、この例で7文字になっているのはこのためだ。

スプライトパターンジェネレータテーブルの変化

アドレス	内容〈2進表記〉	=〈16進表記〉(〈10進表記〉)	
&H3800	00000000	=&H00(0)	パターンデータ
&H3801	00000000	=&H00(0)	
&H3802	00000000	=&H00(0)	
&H3803	00000000	=&H00(0)	
&H3804	00000000	=&H00(0)	
&H3805	00000000	=&H00(0)	
&H3806	00000000	=&H00(0)	
&H3807	00000000	=&H00(0)	

⬇

SPRITE$(0)="<~■□□□~"

⬇

アドレス	内容〈2進表記〉	=〈16進表記〉(〈10進表記〉)	パターンデータをキャラクタに置き換えたもの
&H3800	00111100	=&H3C(60)	<
&H3801	01111110	=&H7E(126)	~
&H3802	11111111	=&HFF(255)	■
&H3803	11011011	=&HDB(219)	□
&H3804	11011011	=&HDB(219)	□
&H3805	11011011	=&HDB(219)	□
&H3806	01111110	=&H7E(126)	~
&H3807	00000000	=&H00(0)	※3

※サンプルに使ったキャラクタは40ページ『とぶんだ!!』の主人公

5つ目のスプライト　スプライトが5つ横にならんだとき、Y座標がまったくおなじなら面番号のもっとも大きいスプライトは表示されない。Y座標がずれていれば、重なっている部分だけについて同様のことが起こる。つまり、面番号のもっとも大きいスプライトの一部分が欠けてしまうのだ。6つ以上の場合も同様だ。この現象を避けるためにスプライトが5つ以上重なったときにはほかのスプライトの消去と表示をくりかえし結果的にぜんぶのスプライトが見えるようにすることがあるが、そうなるとスプライトがちらつくようになる。市販ソフトでもそういうことはたまにあって、冗談でフラッシュ攻撃と呼ばれる。

&HF923　BASICのワークエリアのひとつ。&HF922とセットで、PRINTできるVRAMのパターン名称テーブルの開始アドレスを格納している。～22のほうはアドレス(16進数)の下位2桁、～23のほうが上位2桁を管理している(つまり上位、下位が逆転している)。ここのアドレスをスプライトパターンジェネレータテーブルのアドレスと一致させておくと、PRINTしたときにその文字のキャラクタコードがジェネレータテーブルに書きこまれることになる。ただし、WIDTHをめいっぱいの32にしておかないと、PRINT文では届かない場所ができてしまうので注意。ところで、スプライトパターンジェネレータテーブルは最初0の固まりなのでLOCATE文やカーソルキャラクタなどを使ってPRINTする場所を飛ばせば、そこに0を設定したことになるし、たとえば「月」をPRINTすればそこに1を書きこんだことにもなる。つまりコントロールコードとおなじデータもグラフィックキャラクタで文字に置き換えられるわけだ。ただし、この方法でもあいかわらずDELキーのコード127だけは自由にならない。

VRAMから見たスプライトの仕組み

■面番号とパターン番号

スプライト面番号とスプライトパターン番号は混同されやすいが、BASICの仕組み自体が、この2つを混同しやすいように意地悪しているようだ。

PUTSPRITE命令は、おしりにくっつくカラーコードとパターン番号を省略できる。カラーコードは省略すると白になるのでさすがに入れることが多いが、最後のパターン番号はかなりの確率で省略される(29ページのサンプルプログラムがまさにそれだ)。じっさい、最初に指定する面番号の数値がそのままパターン番号の数値になってしまうのだから、かんたんなプログラムでは指定する必要のないことが多いのだ。

便利ではあるが、そのために面番号とパターン番号が混同されてしまうのだろう。

また、スプライト面ということばは、概念自体がピンとこないために、ついついわかりやすいパターン番号のほうで考えてしまうということもあるだろう。

■スプライト面の実体とは

スプライトの説明でよく出てくる表現は、「画面のまえに0番から31番のスプライト面が重ねられていて、1枚のスプライト面には1つのスプライトしか表示できない」といったたぐいのものだ。これは、アニメで背景にセル画をかぶせて撮影するのとおなじようなイメージなのだが、肝心のスプライト面の実体が見えてこないのでもうひとつピンとこないのかもしれない。

スプライト面の実体はなにか。それは、スプライト属性テーブルだ。属性テーブルは、&H1B00～&H1B03がスプライト面0、&H1B04～&H1B07がスプライト面1というふうに面番号の順にならんでいる。逆にいえば、面番号とは、属性テーブルのアドレスを指定するパラメータなのだ。

面番号と属性テーブルのアドレスとの変換式は次のようになる(SCREEN1の場合)。
アドレス=&H1B00+スプライト面番号×4

このアドレスから1バイトずつ、そのスプライトの〈Y座標〉、〈X座標〉、〈パターン番号〉、〈カラーコード〉の順にデータが収められているのだ。つまり、たとえばパターン番号の場合は、上の式で計算されたアドレス+2のところにある。

あらかじめ1つスプライトを表示しておいて、VPOKEとこの変換式を使っててきとうに書き換えてみると、属性テーブルの働きが見えてくるだろう。

■パターン番号とアドレス

ついでにいうと、パターン番号とスプライトパターンジェネレータテーブルのアドレスの変換式は、8×8ドットの場合、
アドレス=&H3800+パターン番号×8
16×16ドットの場合、
アドレス=&H3800+パターン番号×32
となる。これもそれぞれパターン番号順に8バイト(または32バイト)ごとにならんでいる。

面番号の大きいスプライトは面番号の小さいスプライトに隠れるということ、横に5つ以上のスプライトをならべると面番号のいちばん大きなスプライトが消えたり欠けたりするということ、もうひとつ、16×16ドットのスプライトパターン定義の方法をおぼえれば、それでスプライト表示に関する基本は十分といえるだろう。

ただ珍しいだけのサンプルプログラム

PRINT文でパターンを定義する

```
10 SCREEN 1,1:WIDTH 32
20 POKE &HF923,&H38
30 PRINT "<Bさ「「しA>"
40 POKE &HF923,&H18
50 PUTSPRITE 0,(120,100)
```

↑8×8ドットの拡大モードで渦巻きパターンを定義して表示している

メインRAMの&HF923を行20のように書き換えると、スプライトパターンジェネレータテーブルにパターン名称テーブルが重なる。そこで文字をPRINTすると、文字のキャラクタコードに応じたパターンが設定されるのだ。これならばデータの1～31も「月」などの文字に置き換えることができるのだ。ただし、かならず行40のようにもとにもどしておくこと。これについては前号の「PRINTできるPNT」を参照。

無限にうごめく文字のスプライト

```
10 SCREEN 1;0:COLOR 0,0,0
20 VDP(6)=0:VDP(5)=48
30 PRINT CHR$(RND(1)*224+32);
40 VDP(1)=VDP(1) XOR RND(1)*3
50 GOTO 30
```

↑うごめくスプライト。行10のCOLOR文を文字が見えるように変えてみよう

行20のVDP(6)=0は、スプライトパターンジェネレータテーブルを文字のパターンジェネレータテーブルに重ね、VDP(5)=48はスプライト属性テーブルをパターン名称テーブルに重ねている。すると、てきとうに文字を表示するだけで(文字の色が透明になっているので見えない)たくさんの文字型スプライトがうごめくのだ。行40はランダムにスプライトサイズと拡大・非拡大モードを切り換えているのだ。

16×16の大型スプライトの場合

大型スプライトを定義して表示する

```
10 SCREEN 1,3
20 FOR I=0 TO 15
30 READ A$
40 SL$=SL$+CHR$(VAL("&B"+LEFT$(A$,8)))
50 SR$=SR$+CHR$(VAL("&B"+RIGHT$(A$,8)))
60 NEXT
70 SPRITE$(0)=SL$+SR$
80 PUTSPRITE 0,(120,100)
100 DATA 0000011111100000
110 DATA 0001100000011000
120 DATA 0011000000001100
130 DATA 0110001000000110
140 DATA 0100010000000010
150 DATA 1000100000000001
160 DATA 1000000000000001
170 DATA 1001000000000001
180 DATA 1001000000000001
190 DATA 1000000000010001
200 DATA 1000000000010001
210 DATA 0100000000100010
220 DATA 0110000000000110
230 DATA 0011000000001100
240 DATA 0001100000011000
250 DATA 0000011111100000
```

16進データを使っておなじことをする

```
10 SCREEN 1,3
20 READ A$
30 FOR I=0 TO 31
40 SP$=SP$+CHR$(VAL("&H"+MID$(A$,I*2+1,2
)))
50 NEXT
60 SPRITE$(0)=SP$
70 PUTSPRITE 0,(120,100),15
100 DATA 0718306244888090908080406030180
7E0180C0602010101011111122060C18E0
```

↑左のプログラムを実行した直後の画面

スプライトパターンジェネレータテーブルの状態

アドレス	データ	部分
&H3800	00000111=&H07(7)	左上
&H3801	00011000=&H18(24)	左上
&H3802	00110000=&H30(48)	左上
&H3803	01100010=&H62(98)	左上
&H3804	01000100=&H44(68)	左上
&H3805	10001000=&H88(136)	左上
&H3806	10000000=&H80(128)	左上
&H3807	10010000=&H90(144)	左上
&H3808	10010000=&H90(144)	左下
&H3809	10000000=&H80(128)	左下
&H380A	10000000=&H80(128)	左下
&H380B	01000000=&H40(64)	左下
&H380C	01100000=&H60(96)	左下
&H380D	00110000=&H30(48)	左下
&H380E	00011000=&H18(24)	左下
&H380F	00000111=&H07(7)	左下
&H3810	11100000=&HE0(224)	右上
&H3811	00011000=&H18(24)	右上
&H3812	00001100=&H0C(12)	右上
&H3813	00000110=&H06(6)	右上
&H3814	00000010=&H02(2)	右上
&H3815	00000001=&H01(1)	右上
&H3816	00000001=&H01(1)	右上
&H3817	00000001=&H01(1)	右上
&H3818	00000001=&H01(1)	右下
&H3819	00010001=&H11(17)	右下
&H381A	00010001=&H11(17)	右下
&H381B	00100010=&H22(34)	右下
&H381C	00000110=&H06(6)	右下
&H381D	00001100=&H0C(12)	右下
&H381E	00011000=&H18(24)	右下
&H381F	11100000=&HE0(224)	右下

パターン番号0のデータ

→16×16ドットではパターン番号1つにつき32バイトのエリアが割りあてられる

スプライトには4タイプの大きさがある。8×8ドット(標準サイズ)にノーマルと拡大の2タイプがあり、16×16ドットのスプライトサイズでも同様に2タイプがある。29ページのサンプルプログラムでは、8×8ドットの拡大モードに設定してある。拡大モードというのは、ご存じのとおり、パターンの1ドットが画面の4ドットに対応するモードだ。

このタイプは、SCREEN文の第2パラメータで指定される。そこに0または1を指定すれば、標準サイズの2タイプ、2または3なら16×16ドットの2タイプが指定される(1、3が拡大モード)。

8×8ドットのパターン定義は29ページのとおりだが、16×16ドットの場合はちょっとややこしい。

たとえば、『BUBBLE』のバブルスは16×16ドットのスプライトだ。すなおに2進数でバブルスのパターンを再現してみると、いちばん上のサンプルプログラムのようになる。薄目にして見ればDATAの行にバブルスが見えるだろう。このデータをそのままCHR$に入れるわけにはいかない。CHR$には2進数8桁までの数値しか入れられないし、そもそも、パターンは8ドット単位(8ビット=1バイト単位)でしか定義できないのだ。

そこで、16ドットのパターンを左右2つにわけ、左側と右側を16バイトずつ定義することになる。

左上のプログラムでは、文字関数を使って2進データを左と右の2つに分けて別々に連結しているが、その下のプログラムではVRAMでの状態を最初から意識して16進データを作り、それを機械的に32バイトぶん連結している。

※サンプルに使ったキャラクタは41ページ『BUBBLE』のバブルスくん

X=X+(S=7)−(S=3)の謎を解明する

BASICテクニック

#5 答えが0と−1しかない関係式

BASICの計算式は数学とよく似ているが、おなじだと思っていると突然裏切られる。なかでも大小の関係に応じて答えを出す関係式はとりわけ奇妙だ。

関係式 2つの式や数値、文字の大小関係を「調べる」式のことを一般に関係式という。文字の場合は、キャラクタコードで大小を判断する。大小関係には、右ページの表にあるとおり、大きい、小さい、等しい、大きいか等しい(以上)、小さいか等しい(以下)の5種類がある。ここで、たいせつなのは、関係式が大小の関係を「表す」のではなく「調べる」ためのものだという点。算数で出てくる関係式のようなものは、大小関係を「表す」もので、たとえば「2>1」と書くと、「2が1より大きい」という意味になった。ところが、BASICの関係式としての「2>1」は「2は1より大きいですか」という質問になる。BASICは、それに対して「はい」のかわりに「−1」、「いいえ」のかわりに「0」という値をその式に持たせる働きをする。この働きを関係演算といい、そのための>や=などの記号を関係演算子という。ちなみに、関係演算子は、+、−、＊などの算術演算子よりも優先順位は低く、AND、ORなどの論理演算子よりも優先順位は高い。

STICK関数 念のためにつけくわえておくと、STICK(0)という関数は、カーソルキーの押された方向に応じて0(なにも押されていない)、1(上方向)〜8(左上方向)という値になる。右ページの図の矢印のなかの数値はSTICK(0)がそれぞれの場合に取る値を示している。また、STICK(1)ならジョイスティックのポート1、STICK(2)ならジョイスティックのポート2に対してまったくおなじように働く。

問1 2>1は、いくつになりますか?

BASICの世界に小学校があれば、きっとこんな算数の問題も出ることだろう。

問1 2>1は、いくつになりますか?

答えは、−1だ。

うそだと思う人は試してみよう。念のためにこの式をカッコでくくって、PRINTしてみればいいのだ。

PRINT (2>1)⏎

すると、−1と表示される。この場合、左辺の「2」は、3でも4でも、1より大きくさえあればなんでもいい。この妙な性質の式の名前は関係式という。

また、「1>2」という式なら答えは0になる。関係式の答えはこの、−1か0の2種類しかない。

関係式で使われる演算子(計算記号)は、右ページのように6種類あるが、≠を<>と書いたりする点を除けば、この演算子そのものは小学校の算数で習うものとおなじだ。しかし、BASICの世界では、この式が答えを出す(値を持つ)。上の例からもわかるように、その関係が成立すれば−1、しなければ0という値になる。

いちばんわかりやすい関係式の使われ方はIF文のなかだ。

IF A>B THEN ① ELSE ②

この場合、A>Bが成り立てば①を実行し、成り立たなければ②を実行する。このことはだれもがよく知っているだろうが、じつは、このIF文自体が、関係式の値を利用しているのだ。

IF文は、IFのあとに続く式や数値が−1(ほんとうは0以外の値ならなんでもいい)ならTHEN以下を実行し、0ならELSE以下を実行する性質を持っていて、関係式の性質と相性がいい。もちろん、そうなるように作られたわけだが。

⬆右ページのプログラムを実行するとカーソルキーでスプライトが動く

条件に応じて値を変えるという点では、関係式自体がIF文的働きを持っているわけで、かんたんな座標計算などでは、IF文をまったく使わなくてもプログラムが組める。右ページにあるのは、そのサンプルと仕組みの図解だ。

困ったことに、この関係式の奇妙で便利な性質に関しては、BASICマニュアルにほとんど書かれていないので、知らなかった人もわりと多いのではないだろうか。

関係演算とスティック入力

演算子	意　　味	関係式	3つの場合の関係式の値		
			A＜Bのとき	A＞Bのとき	A＝Bのとき
＜	小さい	A＜B	−1	0	0
＞	大きい	A＞B	0	−1	0
＝	等しい	A＝B	0	0	−1
＜＞※	等しくない	A＜＞B	−1	−1	0
＜＝※	小さいか等しい	A＜＝B	−1	0	−1
＞＝※	大きいか等しい	A＞＝B	0	−1	−1

■関係演算子の意味と関係式の値

※「等しくない」、「小さいか等しい」、「大きいか等しい」の演算子は、それぞれ、「＞＜」、「＝＜」、「＝＞」と逆にならべてもかまわないし、意味もまったくおなじになる。

関係演算を利用した移動計算

```
10 SCREEN1,1:KEYOFF
20 SPRITE$(Ø)="A>IIII>A"
30 S=STICK(Ø)
40 X=X+(S=7)-(S=3)
50 Y=Y+(S=1)-(S=5)
60 PUTSPRITE Ø,(X*8,Y*8-1),8
70 GOTO 3Ø
```

左のプログラムをIF文で翻訳すると

```
10 SCREEN1,1:KEYOFF
20 SPRITE$(Ø)="A>IIII>A"
30 S=STICK(Ø)
40 IF S=7 THEN X=X-1
50 IF S=3 THEN X=X+1
60 IF S=1 THEN Y=Y-1
70 IF S=5 THEN Y=Y+1
80 PUTSPRITE Ø,(X*8,Y*8-1),8
90 GOTO 3Ø
```


上のプログラムはどちらもカーソルキーでスプライトを４方向に移動させるプログラムで、変数SにSTICK(０)の値を代入して、それに応じた座標計算をしている。左のプログラムの行40、50は、右のプログラムの行40〜70とまったくおなじことをやっているが、見ただけではなにをやっているのかわかりにくいだろう。

これは変数Sが取る値を場合にわけて考えるとよくわかる。それを図解したのが、右の図だ。

たとえば、Sが１(上方向)の場合、(S＝１)が−１になって、行50の式はけっきょくY＝Y−１とおなじことになるのだ。

式のなかにさりげなく組みこまれたIF文

関係式を使ってプログラムを組むとなんだかトクした気分になる。まえのページの右側のプログラムのように、ＩＦ文を4つも使って書かなくてはいけないプログラムを、X、Yそれぞれの計算式だけですませてしまえるからだ。じっさい、プログラムは多少短くなるので、たとえばファンダムの1画面タイプなどでは、この関係式を使ったものがじつに多い。

■複雑な判断が必要な場合

ただ、まえのページの例はかんたんな説明ですむように4方向だけにしてあるし、スプライトが画面の外に出てしまうのもまったくコントロールしていない。条件が複雑になってくると、さすがにＩＦ文を使わないとできないのではないかと思うかもしれないが、少なくとも座標計算にかぎっていえば、ＩＦ文でできるものはかならず関係式だけでできる。

斜め入力を受け付け、画面の外に出ないようにするための条件を考えてみよう。

①Sが1のとき→Y＝Y－1
②Sが2のとき→X＝X＋1、Y＝Y－1
③Sが3のとき→X＝X＋1
④Sが4のとき→X＝X＋1、Y＝Y＋1
⑤Sが5のとき→Y＋Y＋1
⑥Sが6のとき→X＝X－1、Y＝Y＋1
⑦Sが7のとき→X＝X－1
⑧Sが8のとき→X＝X－1、Y＝Y－1
⑨Xが0(以下)のとき→X＝X－1をおこなわない
⑩Xが30(以上)のとき→X＝X＋1をおこなわない
⑪Yが0(以下)のとき→Y＝Y－1をおこなわない
⑫Yが22(以上)のとき→Y＝Y＋1をおこなわない

⑨～⑫は、逆に考えたほうがプログラムにしやすいので、次のように考えなおす。

⑨X＝X－1はX＞0のときだけおこなう
⑩X＝X＋1はX＜30のときだけおこなう
⑪Y＝Y－1はY＞0のときだけおこなう
⑫Y＝Y＋1はY＜22のときだけおこなう

①～⑫を、X＝X＋1などの計算をおこなう条件ごとにまとめてみると、

①S＝6、7、8でX＞0→X＝X－1
②S＝2、3、4でX＜30→X＝X＋1
③S＝1、2、8でY＞0→Y＝Y－1
④S＝4、5、6でY＜22→Y＝Y＋1

となる。さらにいいかえると、

①S＞5かつX＞0→X＝X－1(Sは9以上になることがないので、Sの上限については考えなくていい)
②S＞1かつS＜5かつX＜30→X＝X－1
③「S＞0かつS＜3またはS＝8」かつY＞0→Y＝Y－1
④S＞3かつS＜7かつY＜22→Y＝Y＋1

ここまでの考え方の道筋は、ＩＦ文で組むのも関係式だけで組むのもおなじだ。①～④に書かれている文章は、そのまま、ＩＦ文でのプログラムに翻訳することができる。各文章の頭に「ＩＦ」を置き、「かつ」を「ＡＮＤ」、「または」を「OR」、「→」を「THEN」、カギカッコをふつうのカッコに置き換えればそのままプログラムになってしまう。

ＩＦ文を使わないプログラムにするには、たとえば①と②をまとめて、

X＝X＋(S＞5　AND　X＞Ø)－(S＞1　AND　S＜5　AND　X＜3Ø)

という式を作ればいい。また、

X＝X－(S＞5)＊(X＞Ø)－(S＞1)＊(S＜5)＊(X＜3Ø)

というふうに論理演算子の「AND」を「＊」に置き換えたものもよく見かける。右ページのプログラムも、この形にしてある。ただし、このように算術演算子を使う場合、いくつかの関係式をかけあわせるとき、全部が成立したときの値が、偶数個の関係式なら＋1、奇数個なら－1になることに注意しなくてはいけない。

また、ORを算術演算子で置き換えるときは、＋か－を使うことになる。＋を使うか、－を使うかはまわりの＋－の状況によって変わる(右ページ上のプログラム行50参照)。

■関係式の効用の代表例

こうしたＩＦ文とおなじ機能を持った関係式によるプログラムは、ほかにもいろんな可能性を秘めている。

たとえば、右ページの下のサンプルプログラム(いちおうかんたんなゲームになっている)の行40は、関係式の威力を示す代表的な例だ。この短い計算式だけで、獲得したスコアがハイスコアになったかどうかを判断し、もしそうであればハイスコアを更新するというところまでやってのけているのだ。

↑右のゲームの実行画面。ふいにビープ音が鳴り、画面が赤くなる

↑スペースキーをすばやく押すと得点。おくれるとゲームオーバー

論理演算子　AND、OR、XOR、EQV、IMPの5種類があるが、よく使われるのは、AND、OR、XORの3つ。なかでも、AND、ORはIF文のなかで関係式と組み合わされた形でよく出てくる。たとえば、①、②の2つの関係式があるとすると、「①　AND　②」という論理式は、①と②が同時に成り立つ場合(つまり、①も②も－1の場合)のみ、－1という値を持ち、それ以外では0になる。これは日本語に訳すと「①が成り立ち、かつ、②が成り立ちますか」という質問だと考えられる。その質問に対して「はい」のかわりに－1、「いいえ」のかわりに0という値をその論理式に持たせるのだ。同様に、ORの場合も、「①が成り立つか、または、②が成り立ちますか」という質問として考えられる。①、②のどちらか一方(または両方)が成り立つとき、「はい」のかわりに－1、どちらも成り立たないとき、「いいえ」のかわりに0を式の値として持たせるわけだ。この働きを論理演算という。ただし、これは、－1と0のどちらかの値しか持たない関係式と組み合わせた場合のみについていえることで、じっさいには、「3　AND　6」などという計算をさせると、2という答えを出すので、ちょっとやっかいだ。この論理演算子については、なにかの機会に扱うだろう。

論理演算子と算術演算子　「AND」と「＊」は関係式と組み合わせるかぎりにおいてよく似ている。たとえば、①、②の関係式があるとすると、「①AND②」は、①②のどちらも－1のときだけ－1になり、そのほかの場合は0になる。一方、「①＊②」は、①②のどちらも－1のときに1(－1ではない)になり、そのほかの場合は0になる。関係式が3つの場合、「①AND②AND③」は①②③のすべてが－1のときのみ、－1になり、そのほかの場合は0。「①＊②＊③」は、①②③のすべてが－1のときのみ、－1になり、そのほかの場合は0。ANDが＊とちがうのは、すべての関係式が成り立つとき、ANDの式はつねに－1になるのに対して、＊の式は、関係式の個数に応じて1になったり、－1になったりする点だ。これは、マイナスを偶数回かけるとプラスになるからだが、ここさえまちがわなければ、かんたんにANDと＊は置き換えることができる。「OR」と「＋」「－」の関係もこれと似ている。注意しなくてはいけないのは、「①OR②」でいうと、①②のどちらもが成り立つような場合はうまくいかないということだ。なぜなら、両方が成り立つとき「①OR②」はやはり－1になるが、「①＋②」は－2になるし、「①－②」は0になってしまうからだ。ここの例でいえば、ORの左側(2つの関係式とAND)と、右側はまったく別々の場合なのでその心配がなかった。もし、ORの両側が成り立つ可能性のある場合は、SGN関数(その数がマイナスなら－1、0なら0、プラスなら＋1になる)を使って、－1でも－2でもかまわないようにするなど、ひとくふう必要になる。じつは、XORという論理演算子(どちらか一方だけが成り立つときのみ－1で、それ以外は0)は、そのまま、きれいに＋や－に置き換えられる。

複雑なタイプ

斜め入力も受けつけ、画面外にも出ない

```
10 SCREEN1,1:KEYOFF
20 SPRITE$(0)="A>IIII>A"
30 S=STICK(0)
40 X=X-(S>5)*(X>0)-(S>1)*(S<5)*(X<30)
50 Y=Y+((S>0)*(S<3)-(S=8))*(Y>0)-(S>3)*(
S<7)*(Y<22)
60 PUTSPRITE 0,(X*8,Y*8-1),8
70 GOTO 30
```

↑左のプログラムを走らせた画面

X=X-(S>5)*(X>0)-(S>1)*(S<5)*(X<30)

STICK関数の値Sが、6(左下)、7(左)、8(左上)であれば「S>5」の値は−1になる。また、スプライトが画面の左端に来ていなければ、「X>0」も−1になる。つまり、画面の左端以外で左(斜め)方向のキーが押されていれば、この部分は−(−1)×(−1)で−1になり、それ以外では0になる。

同様に、Sが2(右上)、3(右)、4(右下)のときは、「S>1」と「S<5」はどちらも−1になる。また、スプライトが画面の右端に来ていなければ「X<30」も−1になる。つまり、画面の右端以外で右(斜め)方向のキーが押されていれば、この部分は−(−1)×(−1)×(−1)で+1になり、それ以外では0になる。

Y=Y+((S>0)*(S<3)-(S=8))*(Y>0)-(S>3)*(S<7)*(Y<22)

Sが1(上)、2(右上)のとき、「(S>0)*(S<3)−(S=8)」は、(−1)×(−1)−0で1、Sが8(左上)のときも、0×0−(−1)で1。画面の上端でなければ、「Y>0」は−1。つまり、画面の上端以外で上(斜め)方向のキーが押されていれば、+(1)×(−1)、つまり−1になり、それ以外では0になる。

Sが4(右下)、5(下)、6(左下)のときは、「S>3」と「S<7」はどちらも−1になる。また、スプライトが画面の下端まで来ていなければ、「Y<22」も−1になる。つまり、画面の下端以外で下(斜め)方向のキーが押されていれば、全体に−(−1)×(−1)×(−1)で+1になり、それ以外では0になる。

29ページのプログラムとちがって、このプログラムは8方向の入力を受け付け、しかも、スプライトが画面の端で止まるようになっている。そのための判断と計算をやっているのが行40、50で、それぞれの主要部分を解説すると左のようになる。ちょっと複雑だが、落ち着いて考えれば、そうむずかしいことではない。たかだか−1か0かの組み合わせの問題なのだ。

ちなみに、X座標の上限が30、Y座標の上限が22になっているのは、ここで使っているスプライトが8×8ドットの拡大モードになっているために、2文字ぶんの幅を持っているからだ。

世界一短いハイスコアルーチン

ちょっとしたゲームほど、ハイスコア機能があるかないかで、おもしろさもずいぶん変わってくるものだ。下のプログラムの行40に、たぶん世界一短いと思われるハイスコア機能を持つ計算式がある。変数Sはそのときのスコア、変数Hはハイスコアを表す。

なぜ、行40の式で、スコアがハイスコアかどうかの判断をしたうえに、そうだった場合にハイスコアを更新するようなことができるのかは、右の流れ図で解説してある。

ゲームプログラムを作っている読者は、ぜひ、このハイスコアルーチンを使って、自作のゲームにハイスコア機能を入れてほしいものだ。

ハイスコア機能がわかりやすいように、下のプログラムは、いちおうゲームになっている。右下の遊び方をよく読んで遊んでみてほしい。

H=S-(H>S)*(H-S)

H>Sのとき (スコアがハイスコアでない)	S≧Hのとき (スコアがハイスコアになった)
H=S-(-1)×(H-S)	H=S-(0)×(H-S)
H=S+H-S	H=S-0
H=H	H=S
ハイスコア変わらず	現在のスコアがハイスコア

ついでに反射神経もきたえられる

```
10 SCREEN0:KEYOFF:COLOR 15,4
20 FOR I=0 TO RND(1)*540+60:NEXT:PRINT S
30 BEEP:COLOR,8:FOR I=0 TO 40:IF STRIG(0
) THEN 60 ELSE NEXT
40 H=S-(H>S)*(H-S)
50 PLAY"T180L8V15S6M2000O7AAAAA":PRINT"HI
-SCORE="H" SCORE="S:COLOR,4:FOR I=0 TO 1
:I=-(INKEY$=CHR$(27)):NEXT:S=0:GOTO 10
60 FOR I=0 TO 50:IF STRIG(0)*(I>30) THEN
 40 ELSE NEXT:S=S+1:COLOR,4:GOTO20
```

【ゲームの遊び方】RUNすると、まず青い画面になる。画面左上に「0」(これがスコア)と表示されると同時にビープ音が鳴り、画面が赤くなったらすぐに(約0.07秒以内)スペースキーを押す。押したら0.05秒以内にはなす。成功すれば画面はまた青にもどり、スコアが1点増える(スコアは増えるたびに次々に下に表示されていく)。0.1秒〜1秒後、ふたたびビープ音が鳴り、画面が赤くなるので同様にくりかえす。間隔は一定でないので純粋な反射神経が必要だ。リプレイはESC。

論理演算子の典型的な使い道サンプル

BASICテクニック

＃6　パターンを変形するビット演算

論理演算子　今回扱った３つの論理演算子以外に、NOT、IMP、EQVの３つがある。それぞれ特徴的な真理表を持つのでマニュアルで調べよう。
ビット　bit。もともと、binary digit（２進数）の略。コンピュータ関係の入門書などでは、「情報の単位」などとわけのわからないことが書かれているが、むしろ、２進数何桁ぶんかを表す単位と考えたほうがわかりやすいだろう（そのことがそのまま「情報量」に通じる）。たとえば、パターンデータの１行ぶんは、２進数８桁になっているので、８ビットの情報だ、などと使う。しかし、こんな使い方は、BASICではほとんどしない。BASICでビットというときは、たいてい２進数の１桁を指していうことが多い。また、２進数がいくつかならんだものを、ビット列という。このビット列に対して、１桁単位で演算していくことをビット演算という。

■ORの真理表

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

ORやANDはなんのためにあるか

ＢＡＳＩＣの演算子には、＋や－、＊、／以外にも、指数計算（2の3乗などの計算）に使う「^」、整数で割った余りを出す「MOD」、商を出す「¥」、「＜」などの関係演算子、そして「OR」「AND」などの論理演算子がある。なかでも、ふつうの算数とかけはなれて違うのが、論理演算子だ。
ただ、論理演算というもの自体は、その名前がいかにもむずかしそうなわりに、日常的によく使われるものなのだ。
「もし、明日が晴れで、ぼくが暇なら、野球を見に行く」
という文章があるとすると、ＢＡＳＩＣの文法に従って、
ＩＦ〈明日が晴れ〉ＡＮＤ〈ぼくが暇〉ＴＨＥＮ〈野球を見に行く〉
というふうに置き換えることができる。これは、ＩＦに続く２つの要素の両方が成立したときにだけＴＨＥＮ以下のことが起こるという意味になる。ここでは、「明日が晴れで」の「で」がＡＮＤに相当しているのだ。
実際のプログラムでは、
ＩＦ〈カーソルキーの左が押されている〉ＡＮＤ〈キャラクタのＸ座標が０より大きい〉ＴＨＥＮ〈キャラクタが左に動く〉
などという場面でよく出てくる。
こういうＩＦ文のなかに出てくる論理演算はわかりやすいが、ほかにもORやANDは通常の計算式のなかに姿を見せることがある。そのうちのほとんどは、論理演算子のもうひとつの使い道、ビット演算だ。とくに、文字パターンを変形させるときなどによく出てくる。
ビット演算とは、２進数の形にした数値を、各桁ごとに、その論理演算子の真理表に従って演算していくものだ。論理演算を見つけたら、その演算子の両側にある数値を２進数の形にして考えてみれば、なにをしたいのかが目に見える形になる。

●ORを使った太文字

⇧右ページのプログラムを実行すると、「Ａ」のパターンが横に１ドットぶん太くなる

文字を太くする2種類のビット演算

大文字のAを横に太くする

```
10 SCREEN 1
20 AD=ASC("A")*8
30 FOR I=0 TO 7
40 PD=VPEEK(AD+I):PD=PD OR PD¥2
50 VPOKE AD+I,PD
60 NEXT
```

左は、BASICピクニックの#1でやった代表的な変体文字「太文字」のごくかんたんなプログラムだ。変数ADには、SCREEN1のパターンジェネレータテーブルのうち「A」のパターンがある部分の先頭アドレスが入り、行30〜60で、その部分を書き換えている。

変数PDには、8バイトあるパターンデータのうちの1バイトずつが入り、それぞれに対して「PD OR PD¥2」という計算をおこなってから、もとあったところに結果を書きこむ仕組みだ(まんなかの表は上から順に1バイトごとのパターンデータと計算結果を表している)。この計算式には下で解説しているような2つのビット演算が関わっているのだ。

PD	PD¥2	PD OR PD¥2
0 0 1 0 0 0 0 0	0 0 0 1 0 0 0 0	0 0 1 1 0 0 0 0
0 1 0 1 0 0 0 0	0 0 1 0 1 0 0 0	0 1 1 1 1 0 0 0
1 0 0 0 1 0 0 0	0 1 0 0 0 1 0 0	1 1 0 0 1 1 0 0
1 0 0 0 1 0 0 0	0 1 0 0 0 1 0 0	1 1 0 0 1 1 0 0
1 1 1 1 1 0 0 0	0 1 1 1 1 1 0 0	1 1 1 1 1 1 0 0
1 0 0 0 1 0 0 0	0 1 0 0 0 1 0 0	1 1 0 0 1 1 0 0
1 0 0 0 1 0 0 0	0 1 0 0 0 1 0 0	1 1 0 0 1 1 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

■図1　太文字に使われるビット演算の仕組み

0	0	1	1	0	0	0	0
0	1	1	1	1	0	0	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0
1	1	1	1	1	1	0	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0
0	0	0	0	0	0	0	0

2で割るとビット列は右に1桁ずれる

●もとのビット列

1	0	0	0	1	0	0	0

¥2

●演算結果

0	1	0	0	0	1	0	0

ORでつなぐと2つのビット列が重なる

●もとのビット列

1	0	0	0	1	0	0	0

●OR演算の相手

0	1	0	0	0	1	0	0

●演算結果

1	1	0	0	1	1	0	0

【2で割る】「¥」は算術演算子の1つで、商(割り算した結果の小数点以下を切り捨てたもの)を求めるものだが、とくにビット演算では「¥2」という形でよく現れる。ビット列は2進数なので、2で割ると1桁小さくなるのだ。これは10進数を10で割ると、1桁小さくなるのとまったくおなじことだ。1桁小さくなるということは、0、1のパターンが右に1桁ずれるということで、これはそのままパターンを右に1ドットぶんずらすことになる。同様に、「¥4」(4は2の2乗、つまり、10進数でいえば100にあたる)とすると2ドットぶん右にずれる。ちなみに、「¥」のかわりに「/」を使ってもおなじことだが、わずかだが「¥」のほうが計算が速いのだ。ついでにいうと、「*2」とするとビット列は反対に左にずれるが、数値を2進数8桁内に収めるため、「AND255」を加えなければならない。

【OR演算】これは、パターンを変形させるときにはいろんな場面で出てくるビット演算の基本ともいえる演算だ。ORという論理演算は、0、1のビットに対して左ページにあるような演算結果を出す。0と0なら0だが、それ以外なら(つまり演算対称のどちらかに1があれば)結果は1になる。これは、パターンで見ると、ちょうど重ねあわせたのとおなじことになる。上の太文字処理でいえば、通常の「A」のパターンと、2で割って右に1ドットずらした「A」のパターンを重ねあわせているのが、このORだ。上のプログラムの「PD¥2」を「PD¥4」に変えると、BASICピクニック#1で紹介した「乱視文字」(文字がだぶって見える)のパターンになるし、「PD OR PD¥2 OR PD¥4」と3つ重ねれば、3倍太いボテボテ文字になる(ただし、小文字のmなどは完全につぶれてしまうが)。

XOR　eXclusive OR(エクスクルシブ・オア：排他的ORと訳される)の略。XORも仲間のORも、両辺のどちらかに1があれば演算結果が1になる。違うのは、両辺のどちらにも1があるときだ。ORは1になるが、XORは0になる。右ページの反転処理はこの性質を利用しているのだ。だから、ぜんぶ1のパターン(10進数の255)でなくても、たとえば、右半分4桁が1、左半分4桁が0のパターンでXORすると、右半分だけが反転した文字になる(ただ、見にくくてしょうがないが)。

AND　「そして」や「と」として有名だが、論理演算界では「かつ」と呼ばれている。XORはヒネクレモノという印象があるが、ANDはリチギモノという印象がある。とにかく両辺に1がなければ絶対に1にならないぞという、強く、頑固な意志が感じられ、わたしは彼が好きだ。彼の得意技ビットマスクは、たとえれば「窓」になるだろうか。つまり、1の部分が向こうの見える窓で、0の部分が見えない壁(演算結果が0になる)という感じなのだ。パターンデータ以外の使い道で有名なのは、キーマトリクスを利用した使い方だ。キーマトリクスとは、すべてのキーに対して、押されているか押されていないかの情報を記録しているメモリの一部分のことだ。1つのキーに対して1ビット単位で記録しているのだが、それを8ビットごとにまとめてアドレス1つぶんに収めている。これを利用してキーの入力状態を調べるときにANDが役に立つ。なにしろ、データは1バイト(8ビット)ごとにしか取り出せないので、ANDで必要な桁以外を隠して処理するわけだ。そのビットが0なら、全体が0になり、1なら数値はわからないが0でない数になって、調べやすい。

■XORの真理表

A	B	A XOR B
1	1	0
1	0	1
0	1	1
0	0	0

■ANDの真理表

A	B	A AND B
1	1	1
1	0	0
0	1	0
0	0	0

パターン作成用ビット演算の基礎

まえのページの図を見ると、「ビット演算」などという、いかにもな名前はただのコケおどしで、けっきょく、方眼紙に書いたパターンをずらしたり、重ねたりしているだけの、単純な作業だとわかるだろう。

これは、パターン作成にかぎったことではないが、プログラムで「むずかしい」と思われることは、たいていは単純すぎてピンとこないだけのことが多い。

たとえば、マシン語の1つ1つの命令は、2つの数をたすとか、あるアドレスにある内容をほかの場所に転送するなどといった、きわめて単純なものばかりなのだ。マシン語がむずかしいのは、その単純な命令を組み合わせて複雑な作業をやらせなければいけないからだ。

それに比べて、パターン作成用のビット演算は、0と1をならべてみれば、形も効果もそのまま目に見えるのでかんたんだ。

●XORを使った反転文字

●ANDを使った失恋ハート

↑文字を反転させたり、文字からあるパターンを抜きとったりが、比較的短いプログラムでできる。論理演算子は、こういうビット演算のなかで目に見える働きをしているのだ。

■なぜむずかしく見えるか

原理も作業も単純なパターン作成用のビット演算に、わかりにくい部分があるとすれば、たぶん3つにしぼられるだろう。①VRAMを相手にしなければいけないこと、②右のページにあるプログラムのように、「XOR 255」とか、「AND 247-(I AND 1)*8」といった、見ただけではピンとこない数値や式が出てくること、③パターンデータはかならず2進数8桁でなければならないこと。

①に関しては、BASICピクニックの#1でかなりくわしくやったのでここではくりかえさない。#1を読んでいない人は、ファンダムのプログラムで実例を見ていくか、『MSXポケットバンク⑨グラフィックス秘伝』(アスキー出版局)などを参考にしてほしい。

■たんに短くしているだけ

②は、そこに出ている数値や式を2進数の形にもどしてじっくり見ていけばわかるはずだ。

たとえば、「数字の0を反転させる」の行40にある「255」は2進数の「&B11111111」のことだ。2進数の形になっていると、XORの意味さえ知っていればピンとくるだろう。10進数の形で書かれているのは文字を節約するためだ。

また、「ハートマークを失恋させる」の行40にある「AND 247-(I AND 1)*8」も同様だ。ここでは、次の2つの処理を交互にやろうとしているだけなのだ。

AND &B11110111
AND &B11101111

この2進数は、上が10進数の247、下が239になるので、

AND 247
AND 239

をくりかえしてもおなじ。この作業をさらに1つの式にしたものが行40にある式だ(くわしくはプログラムの右の説明参照)。

けっきょく、2進数の形で書けばよく見えることが、プログラムの節約のためにちょっと見えにくくなっているだけなのだ。

■パターンは2進数8桁だけ

③の問題は、まえのページでもちょっと触れたが、ときどき「AND 255」という処理をする必要があるというていどの問題にすぎない。

VRAMにパターンデータを書きこむときは、そのデータが2進数8桁で表せる数、つまり、0から255までの整数でなければならない。この範囲を超えると、エラーになる。

ところが、たとえばパターンデータを左へずらすために「*2」したり、そのほかの処理を施したときに、パターンデータ候補がこの範囲を超える場合がある。そのために、ある種の計算をするときは、最後に「AND 255」をつけているのだ。255はまえにも出てきたが、1が8桁ならんだ数値のことだ。

この3つのことさえ押さえれば、もうパターン作成のためのビット演算に関してはいうことがほとんどない。

ANDとORの使用例1つずつ

数字のØを反転させる

```
10 SCREEN 1
20 AD=ASC("Ø")*8
30 FOR I=Ø TO 7
40 PD=VPEEK(AD+I) XOR 255
50 VPOKE AD+I,PD
60 NEXT
```

このページのプログラムの基本構造は、37ページのプログラムとおなじ。変数ADには文字のパターンが入っている先頭アドレスが入る。行20のASC関数のなかをべつの文字に換えればその文字が変形する。

論理演算子XORの使い道は、左ページ下の真理表を見ただけではピンとこないだろうが、基本的にはこのプログラムの例のように、パターンの反転(0と1を逆転させる)に使われる。あるビットと1をXORするとそのビットが0なら1、1なら0に変わるわけだ。ここの例では、8桁ぜんぶ1(10進数で255)のビット列でXORしているが、いろんなパターンでXORしてみると、おもしろい形に出会うことができるだろう。

VPEEK(AD+I)	255	VPEEK(AD+I) XOR 255
Ø 1 1 1 Ø Ø Ø Ø	1 1 1 1 1 1 1 1	1 Ø Ø Ø 1 1 1 1
1 Ø Ø Ø 1 Ø Ø Ø	1 1 1 1 1 1 1 1	Ø 1 1 1 Ø 1 1 1
1 Ø Ø 1 1 Ø Ø Ø	1 1 1 1 1 1 1 1	Ø 1 1 Ø Ø 1 1 1
1 Ø 1 Ø 1 Ø Ø Ø	1 1 1 1 1 1 1 1	Ø 1 Ø 1 Ø 1 1 1
1 1 Ø Ø 1 Ø Ø Ø	1 1 1 1 1 1 1 1	Ø Ø 1 1 Ø 1 1 1
1 Ø Ø Ø 1 Ø Ø Ø	1 1 1 1 1 1 1 1	Ø 1 1 1 Ø 1 1 1
Ø 1 1 1 Ø Ø Ø Ø	1 1 1 1 1 1 1 1	1 Ø Ø Ø 1 1 1 1
Ø Ø Ø Ø Ø Ø Ø Ø	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1

■図2 反転文字に使われるビット演算

1	Ø	Ø	Ø	1	1	1	1
Ø	1	1	1	Ø	1	1	1
Ø	1	1	Ø	Ø	1	1	1
Ø	1	Ø	1	Ø	1	1	1
Ø	Ø	1	1	Ø	1	1	1
Ø	1	1	1	Ø	1	1	1
1	Ø	Ø	Ø	1	1	1	1
1	1	1	1	1	1	1	1

ハートマークを失恋させる

```
10 SCREEN 1
20 AD=ASC("♥")*8
30 FOR I=Ø TO 7
40 PD=VPEEK(AD+I) AND 247-(I AND 1)*8
50 VPOKE AD+I,PD
60 NEXT
```

ANDによるビット演算は、とくに「ビットマスク」と呼ばれ、パターンの一部分だけを切り取りたいときに使われる。あるビットパターンに対して、切り取りたい形に0をならべたビット列(ほかの部分はすべて1)をANDすると、その部分にあったビットだけが0になる。逆に見ると、残したい範囲だけに1をならべておくとその部分のパターンだけが残ることになる。

行40のAND以降にある式は、Iが偶数のときは247、奇数のときは239になるが、この数は下図のように2つの切り取りパターンを交互にくりかえしたものだ。I AND 1は、Iが1増えるたびに全体として0と1をくりかえす(これもビットマスクの一種)。これを利用して2つの数を1つの式にまとめているのだ。

VPEEK(AD+I)	247-(I AND 1)*8	VPEEK(AD+I) AND 247-(I AND 1)*8
Ø 1 1 Ø 1 1 Ø Ø	1 1 1 1 Ø 1 1 1	Ø 1 1 Ø Ø 1 Ø Ø
1 1 1 1 1 1 1 Ø	1 1 1 Ø 1 1 1 1	1 1 1 Ø 1 1 1 Ø
1 1 1 1 1 1 1 Ø	1 1 1 1 Ø 1 1 1	1 1 1 1 Ø 1 1 Ø
1 1 1 1 1 1 1 Ø	1 1 1 Ø 1 1 1 1	1 1 1 Ø 1 1 1 Ø
Ø 1 1 1 1 1 Ø Ø	1 1 1 1 Ø 1 1 1	Ø 1 1 1 Ø 1 Ø Ø
Ø Ø 1 1 1 Ø Ø Ø	1 1 1 Ø 1 1 1 1	Ø Ø 1 Ø 1 Ø Ø Ø
Ø Ø Ø 1 Ø Ø Ø Ø	1 1 1 1 Ø 1 1 1	Ø Ø Ø 1 Ø Ø Ø Ø
Ø Ø Ø Ø Ø Ø Ø Ø	1 1 1 Ø 1 1 1 1	Ø Ø Ø Ø Ø Ø Ø Ø

■図3 ビットマスクと呼ばれるビット演算

Ø	1	1	Ø	Ø	1	Ø	Ø
1	1	1	Ø	1	1	1	Ø
1	1	1	1	Ø	1	1	Ø
1	1	1	Ø	1	1	1	Ø
Ø	1	1	1	Ø	1	Ø	Ø
Ø	Ø	1	Ø	1	Ø	Ø	Ø
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

SOUND文を使いこなすための基礎知識

BASICテクニック

#7 PSGのグラスノスチ

SOUND文が使えるようになるには、PSGの構造を知ればいい。PSGは、14個のつまみを持ったシンセサイザーみたいなものなのだ。

PSGのつまみをいじるSOUND文

MSXの電源を入れたあと、ほかになにもしなくても、

SOUND 8, 15↵

とするだけでビープ音とおなじ高さの音が鳴り続ける。

どうしてそうなるのか。PSGの構造を知るまでは不思議だが、知ったあとはごくあたりまえの現象なのだ。

PSGは、右ページの図のように14個のサウンドレジスタというものの複合体だ。

レジスタとは、一般に記憶容量の小さなRAMの一種で、PSGのレジスタの場合、最大で8ビット、つまり1バイトの容量しかない。ふつう使われているメインメモリ(RAM)やVRAMでいえば、1番地ぶんの大きさで、文字なら1文字、数値なら0～255の整数しか記憶できない。

この容量の小さなレジスタは音楽機器のつまみだと考えるとピッタリくる。じっさい、PSGは、たとえではなく、簡易ではあるけれどシンセサイザーの一種なのだ。

そして、PSGというシンセサイザーの14個のつまみをまわすための命令がSOUND文だ。

SOUND文は、

SOUND〈レジスタ番号〉,〈データ〉

という書式で使う。これは指定された番号のレジスタにデータを書きこんでいるだけなのでピンとこないが、各レジスタの機能をはっきり知っていればだいたいの効果が見えてくる。

```
 r 0= 85
 r 1=  0  r 8=  0
 
 r 2=  0            r 7= 56:111000
 r 3=  0  r 9=  0
 
 r 4=  0            r11= 11
 r 5=  0  r10=  0   r12=  0
 
 r 6=  0            r13=  0
Ok
SOUND 8,15
```

↑右ページのプログラムの実行画面。PSGは初期状態。レジスタ8に15を入れると音が出る

PSG　Programable Sound Generator(プログラマブル・サウンド・ジェネレータ)の略。MSXの基本的な音源で、一種のシンセサイザー。PSGの14個のレジスタに直接データを書きこめるのはSOUND文だけだが、PLAY文やBEEP文もこのPSGを使って音を出している。そのために、PLAY文で音が出たり、ビープ音が鳴ったり(これはエラーやCTRL＋Gで鳴ったときもまったくおなじ)すると、PSGの状態が変わる。**【PLAY文の場合】**そのチャンネルで最後に出た音の周波数用データがレジスタ0～5に残る。レジスタ8～10はPLAY文終了とともにすべて0になる。エンベロープ関係の指定があった場合は周期、パターンともレジスタ11～13に残る。**【ビープ音の場合】**レジスタ0は85、レジスタ1は0。レジスタ7は56。レジスタ8～10は0。これはそれぞれ各レジスタの初期値(電源を入れたときの状態)。※MSX2以上の機種でビープ音の種類を変えているときは、種類によってレジスタ0、1の値にちがいがある。とくに、2番のビープ音は3チャンネルとも使用しているため、レジスタ0～5の初期値がそれぞれ順に166、0、167、0、168、0となっている。なお、以上のことはすべて右ページのプログラムで確認することができる。

PSGとCTRL＋STOP　ふつうプログラムの実行をとちゅうで止めるために使うCTRL＋STOPもPSGに影響する。このキーを押すと、レジスタ7～10が、ビープ音と同様に初期値にもどる。

サウンドレジスタをのぞいてみよう

レジスタの値を画面に表示する

```
10 SCREEN 0:WIDTH 40:COLOR15,4,7
20 CLEAR 200,&HD000:DEFUSR=&HD000
30 FOR I=0 TO 8:POKE &HD000+I,VAL("&H"+M
ID$("21F8F77ECD960077C9",I*2+1,2)):NEXT
40 KEY1,CHR$(11)+"GOTO50"+CHR$(5)+CHR$(1
3):KEY2,"SOUND":KEY3,"R%=   :GOSUB80"
50 GOSUB 60:END
60 LOCATE 0,0:FOR K=0 TO 10:PRINT CHR$(2
7);"K":NEXT
70 FOR R%=0 TO 13:GOSUB 80:NEXT:RETURN
80 IF R%<14 THEN V=USR(R%):PRINT CHR$(27
);"Y";MID$(" !!!#!$!&!'!)!#3!*$*'*&3'3)3
",R%*2+1,2);USING"r##=###";R%;V+(R%=7)*1
28;:IF R%=7 THEN PRINT ":";RIGHT$(BIN$(V
),6)
90 LOCATE 0,10:RETURN
```

【プログラムの使い方】●このプログラムを実行すると、画面の上10行ぶんのスペースに各レジスタの番号とその時点での値を10進数で表示する(レジスタ7だけは10進数と2進数の両方で表示している)。●「r」はレジスタの略で、「r 0」はレジスタ0を意味している。●このプログラムの表示は、ほぼ下の図と対応している。各レジスタの機能と関係は、下の図を見てほしい。●表示させたあとでレジスタの中身を書き換えても、表示されている値は変わらない。現在の値を表示させたいときはF1キーを押すとレジスタ表示部分全体を現在の値に書き換えるようになっている。●1つのレジスタだけを現在の値に書き換えたいときは、F3キーを押して「R%=」のあとにそのレジスタの番号を書き入れてリターンキーを押せばいい。●F2キーに「SOUND」も設定されているのでてきとうに利用しよう。●このプログラムはこれだけで動くが、43ページのパーツを加えていっていろいろ実験してみよう。

【注意】行30はマシン語(サウンドレジスタの値を読み出す)を書きこんでいる部分なのでとくに注意して打ちこむこと。また、行80のMID$のなかの文字列(「"」のあとにある空白を見落とさないように)はエスケープシーケンスで位置指定するための文字列なのでここも注意して打ちこむこと。表示される位置が左ページの写真とちがうときは、ここの文字列があやしい。

各レジスタの機能と関係

チャンネルA

レジスタ0 周波数微調整 085 0〜255

※レジスタ8〜10では、16〜31のどれを指定しても効果はまったくおなじ

レジスタ1 周波数粗調整 00 0〜15

レジスタ8 音量:0〜15 00 エンベロープ指定:16〜31

チャンネルB

レジスタ2 周波数微調整 000 0〜255

レジスタ3 周波数粗調整 00 0〜15

レジスタ9 音量:0〜15 00 エンベロープ指定:16〜31

チャンネルC

レジスタ4 周波数微調整 000 0〜255

レジスタ5 周波数粗調整 00 0〜15

レジスタ10 音量:0〜15 00 エンベロープ指定:16〜31

ノイズ

レジスタ6 平均周波数 00 0〜31

※ノイズはチャンネルA〜Cで共通

レジスタ7 56 → 1 1 1 0 0 0
ミキサー 0〜63
ノイズ トーン

ミキサーと呼ばれるレジスタ7

初期値56を2進数にしたものが「111000」だ。ここで0になっているところは音が設定どおり出て、1になっているところはまったく音が出ない。現在は、チャンネルA〜Cのトーンだけが出ることを許されている(ただ、どれも音量が0なので音はしない)。たとえば、チャンネルAのトーンとノイズだけを出したいときは、ここに&B110110、つまり、54を書きこむというふうに先に2進数の形でデータを考えるといい。

エンベロープ

レジスタ11 周期微調整 011 0〜255

レジスタ12 周期粗調整 000 0〜255

レジスタ13 パターン番号 00 0〜15

※エンベロープはチャンネルA〜Cで共通

※この記事で使用しているプログラムは16K以上のRAMが必要です

トーンとノイズ　音程のあるきれいな音のことをトーン、にごった音(いろんな周波数の音が混じると音がにごる)のことをノイズ(雑音)という。
周波数　音は波(振動)の一種で、1秒間にくりかえす振動の回数を周波数という。単位はヘルツ(Hz)。たとえば、楽器のチューニングの基準音となる440ヘルツの音(MMLでいえばO4A、人の産声もこの音だといわれる)は、1秒間に440回振動していることになる。この周波数とまったく逆の関係にあるのが周期(サイクル)だ。
周期　波が1回振動するのにかかる時間のことを周期という。たとえば、1秒間に2回振動する波の周期は、0.5秒になる。基準音のO4Aなら、周期は440分の1秒だ。つまり、周期の単位を秒にした場合は、周期と周波数には次の関係があることになる。周期×周波数＝1。
MML　Music Macro Language(ミュージック・マクロ・ランゲージ：音楽用拡張言語)の略。PLAY文で使う演奏用コマンドのこと。たんにPLAY文データなどともいうが。
エンベロープ周期　エンベロープは音量の時間的変化のパターンを決めるものだ。かならずしもくりかえしのパターンばかりではないが、最初に最小音量から最大音量に変化する(またはその逆)時間のことを波とおなじように「周期」と呼ぶ。エンベロープのパターンは、1回しかそのパターンにならないものと、その周期でくりかえすもの、2倍の周期でくりかえすものがある。くりかえすパターンの場合、ノイズに短い周期のエンベロープをかけると、その周期分の1(または2周期分の1)の周波数を持つ音に聞こえるのだ(これを利用したのが行150)。ノイズだけでなく、その周波数と微妙にずれたトーンにかけた場合も奇妙なうねり効果が出ておもしろい。
エンベロープパターン　エンベロープパターン番号は、MMLのSコマンドで指定する値であり、またレジスタ13に書きこむ値でもある。番号は0～15まであるのにパターンは8種類しかない。これは、0～3と15、4～7と9がそれぞれおなじパターンを共有しているからだ。不思議だがほんとうだ。
うねり　周波数がわずかにちがう2つの音を同時に鳴らすと、2つの周波数が干渉しあって音の響きにうねりが出る。1つ1つは1本調子の音でも、このうねりによって音に奥行きが出てくる。たとえば、MSX2以上の機種のみで使える2番のビープ音はこのうねりを使って奥行きのあるビープ音を出しているのだ(40ページの傍注参照)。この周波数のずれを意識していろんな実験をしてみるとおもしろい。たとえば、レジスタ0と2に6、レジスタ1と3に0、レジスタ8と9に15、レジスタ7に60(A、Bチャンネルのトーンのみオン)を入れてみよう。この音は18キロヘルツ以上の高音で聞こえるとしてもかすかにしか聞こえない。ところが、レジスタ0だけを5(もっと高い音)にすると急に聞こえやすくなる。これは周波数のずれのせいだ、と思う。きっとそうだ、そうにちがいない。ちがうということが証明できる人はBASICピクニックまでお手紙ください。

SOUND文用データの見つけ方

ページが裏表になってやりにくいだろうが必要に応じて41ページの図を見返してほしい。

■各チャンネルごとのレジスタ

●レジスタ0・1(2・3、4・5も同様)

そのチャンネルから出るトーンの周波数を決めるレジスタ。
〈レジスタ0の数値〉＋〈レジスタ1の数値×256〉
が周波数用のデータになっているので、レジスタ0が微調整用、レジスタ1が粗調整用になる。

レジスタ0は1バイトぶん、つまり、0～255の範囲の値を扱えるが、レジスタ1は4ビットしか受けつけないため、0～31の範囲の値しか扱えない。レジスタ1に31以上の値を指定したときは、その値を32で割った余りが書きこまれるようになる。32を指定したときは0、63を指定したときは31になるわけだ。

ところで、このデータはそのまま周波数を表しているのではない。このデータが大きいほど周波数は低くなり(音が低くなる)、小さいほど周波数は高くなる(音が高くなる)ので、注意しておく必要がある。このデータを周波数に変換するための公式が右ページの上にある式だ。

この式を使わなくても、たとえばMMLの「O5C」の音のデータを知りたいときは、
PLAY"O5C"⏎
を実行しておいて、レジスタ0とレジスタ1の値を調べればいい。レジスタ0と1は、PLAY文が実行を終えてもそのまま残っているからだ。この場合、レジスタ0には214、レジスタ1には0が入っているはずだ。

●レジスタ8(9、10も同様)

そのチャンネルから出るトーンおよびノイズの音量調整、またはエンベロープをかけるかかけないかを指定するレジスタ。

0～31の値を扱う(それ以上の値はレジスタ1と同様)が、0～15のときのみ音量調整になり(数値の意味はMMLのVコマンドとまったくおなじ)、16～31の場合はすべてエンベロープの指定になる。

知っていても役には立たないが、MMLのSコマンドでエンベロープパターンを指定すると、このレジスタには〈16＋エンベロープパターン番号〉が書きこまれている(ただしPLAY文の音が鳴り終わるとレジスタ8は0にもどる)。これを逆にして、SOUND文でレジスタ8に30を書きこんだとしても、エンベロープパターンが14になるわけではない。

■各チャンネル共通のレジスタ

●レジスタ6

ノイズの平均周波数用データ。ノイズは、雑多な周波数の集まりなので、その平均値を取って指定する。ここの値によって微妙にノイズの音色が変わる。

扱える数値は0～31の範囲(これを超える範囲の値についてはレジスタ1などと同様)。このデータも、周波数そのものではない。公式はレジスタ0・1のものと似ていて、「r0＋r1×256」のかわりに「r6」を入れるだけでいい。

レジスタ6のデータはチャンネルA～Cに共通で、たとえば3チャンネルともノイズを出すとするとすべておなじ平均周波数のノイズを出すことになる。

●レジスタ7

いわゆるミキサー。各チャンネルから出すトーンとノイズをここで取捨選択する。このレジスタ7の値は、2進数6桁の形にすれば、そのまま意味がわかる。上位3桁が各チャンネルのノイズのスイッチ、下位3桁が各チャンネルのトーンのスイッチで、1のときスイッチオフ、0のときスイッチオンになっていると考えればいいのだ。

●レジスタ11・12

エンベロープ周期を指定するレジスタ。レジスタ0・1の関係に似ていて、
〈レジスタ11の数値〉＋〈レジスタ12の数値×256〉
がエンベロープ周期用のデータになる。どちらのレジスタも0～255の値が扱える。

MMLのMコマンドで指定された数値は、レジスタ11・12の周期用データと完全に一致する。
PLAY"S1M500C"⏎
(最低1つは音名を指定しないとレジスタは変化しない)
として、レジスタの中身を見てみると、レジスタ11は254、レジスタ12は1になっているはずだ(500＝254＋1×256)。

ただし、このデータも周期そのものではなく、右ページの2番目の公式を経て実際の周期となる。エンベロープパターンのなかでもくりかえすパターン(8、10、12、14)では、この周期を計算しておくと、何秒間隔で音が大きくなるかがわかることになる(10番、14番では周期の倍の間隔になることに注意)。

●レジスタ13

エンベロープパターンを指定するレジスタ。0～15の範囲の値が扱え(範囲を超える値についてはほかと同様)、これはそのままエンベロープパターン番号を意味している。MMLのSコマンドで指定する番号と完全に一致し、Sコマンドを実行して音を出すと、このレジスタにSの次の数値が書きこまれる。

また、レジスタ11～13は、レジスタ8～10でエンベロープ指定されているときだけ意味を持ち、エンベロープ指定されているチャンネルはすべておなじエンベロープがかかることになる。

レジスタのそれぞれのデータの意味と関係がわかったら、試行錯誤しながらいろんな値を書きこんで実験してみよう。右ページに周波数、周期計算用の追加プログラムと、サンプルプログラムを用意してあるのでてきとうに利用してほしい。

PSGと遊ぶためのいくつかの道具

440ヘルツの音を出すには

$$周波数 = \frac{111860.78125}{r0+r1\times256}(Hz)$$

```
100 INPUT"Hz";H:A=111860.78125#/H:R1=INT
(A/256):R0=INT(A-R1*256):PRINT"R0=";R0;"
R1=";R1:END
110 INPUT"R0,R1";R0,R1:H=111860.78125#/(
R0+R1*256):PRINT H;"Hz":END
```

エンベロープはパターンと周期で遊ぶ

$$周期 = \frac{256\times(r11+r12\times256)}{1787725}(秒)$$

```
120 INPUT"CYCLE";T:R=T*1787725#/256:R2=I
NT(R/256):R1=INT(R-R2*256):PRINT"R11=";R
1;"R12=";R2:END
130 INPUT"R11,R12";R1,R2:T=256*(R1+R2*25
6)/1787725#:PRINT T;"s":END
```

【エンベロープ周期】上の公式がレジスタ11、12の値とエンベロープ周期の関係を表す。この公式を使った計算用プログラムが行120、130。行120は、設定したい周期(約9.3846秒まで)を入力すればレジスタ11、12の値を表示し、行130では、その逆にレジスタの値から周期を計算する。エンベロープの周期は音の周波数に似た使い方もできる(「ド・デ・ビ・バ……」参照)。

【エンベロープパターン】右はエンベロープパターン番号とパターンの一覧表。対応した番号のパターンで音がうねるわけだ。9番と15番のパターンは一度音を出したらそれっきりウンともスンともいわない。

【プログラムの使い方】このページのプログラムは、41ページのプログラムに追加して使うためのものだが、追加部分は各行とも独立しているので、必要ないと思う行は追加しなくても問題はない。いったん前ページのプログラムの部分を実行したあと、GOTO文で行ごとに使用する。

【周波数】左の公式はレジスタ0、1の値(公式中ではr0、r1と略記)と周波数の関係を表すものだ。この公式を使った計算用プログラムが行100と110。行100では、出したい音の周波数(13.655~111860.78125ヘルツの範囲内)を入力すると各レジスタに入力すべき値を表示する。たとえばこれで440ヘルツの周波数を調べると、「R0=254　R1=0」と表示される。行110はその逆で、レジスタ0、1の値をコンマで続けて入力すると(ただし、両方とも0を入れるとエラーになる)それに対応する音の周波数を表示する。

●エンベロープのパターン番号

レジスタ13の値	パターン (周期)
8	
9、4~7	
10	
11	
12	
13	
14	
15、0~3	

●ピルルルルルルル……

```
140 SOUND 7,56:SOUND 0,85:SOUND 1,0:SOUN
D 8,16:SOUND 13,8:SOUND 12,0:GOSUB 60:FO
R I=0 TO 1:A=A+5-STICK(0) AND 255:SOUND
11,A:R%=11:GOSUB 80:I=-STRIG(0):NEXT:END
```

【現象】行140を追加して、GOTO140を実行すると、「ピルルルル……」と下降しながらビリビリした音を出し、同時にレジスタ11の値をそのたびに書き換える。カーソルキーの下を押すとその時点での状態を保ち、それより右のほうを押すとゆるやかな下降になり、左のほうを押すと逆に上昇する。スペースキーを押すとそのときの音のままプログラムが終了する。【解説】音の高さ自体は変えずに、8番のエンベロープをかけ、レジスタ11を書き換えることで音の高低を作っている。これを応用したのが次のサンプル。

●ド・デ・ビ・バ・ゾ・ダ・ジ・ド

```
150 RESTORE 150:SOUND 8,16:SOUND 7,55:SO
UND 6,0:SOUND 13,12:SOUND 12,0:GOSUB 60:
FOR I=0 TO 7:READ A:SOUND 11,A:R%=11:GOS
UB80:FOR W=0 TO 200:NEXT:NEXT:END:DATA 1
06,95,84,79,71,63,56,53
```

【現象】行150を追加して、GOTO150を実行すると、にごった音でドレミファソラシドを演奏してプログラムを終了する。【解説】「SOUND7,55」でチャンネルAのノイズ以外はスイッチオフにしている(55は&B110111)。12番のエンベロープをかけ、そのエンベロープに乗せてノイズを出しながら、やはりレジスタ11を行末のデータの順に書き換えている。このデータは、O2C~O3Cの音の「周期」(1÷〈周波数〉)。エンベロープの周期をうまく設定すればノイズでも音階を表現できるのだ。

●バラララララ……バラララララ……

```
160 FOR I=0 TO 2:SOUND I*2,166+I:SOUND I
*2+1,2:SOUND I+8,16:NEXT:SOUND 11,0:SOUN
D 12,2:SOUND 13,12:SOUND 6,31:SOUND 7,48
:GOTO 50
```

【現象】行160を追加して、GOTO160を実行すると、音を出しながらすぐに画面上部のレジスタの値を表示しなおし、プログラムが終了する。その後も、微妙に音色を変えながらヘリコプターのローター音に似た音を出し続ける。【解説】ノイズに12番のエンベロープを短めの周期でかけるとヘリコプターのローター音のようになるが、ここではさらにチャンネルA~Cにわずかにちがう周波数を設定して音を出している。このため、トーンの響きがゆったりとしたうねりを持ち、全体として微妙に音色を変えていくわけだ。

意外に奥の深いLINE文の遊び

BASICテクニック

#8 ラインダンスのステップ

グラフィックモード画面の見えない座標のなかをLINE文でさまよってみよう。STEP指定や三角関数やカラーパレットのオプションをつけて。

お遊戯　幼稚園語でダンスのこと。
絶対指定と相対指定　絶対指定とはある点を座標で表すことで、相対指定とは別の基準となる点からどのくらい離れているか(これが増分)で表すこと。相対指定を使うと、基準点を変更するだけで全体の位置をかんたんにずらせるなどなにかと便利。
三角関数　図1のように角Aを持つ直角三角形の場合、斜辺の長さをLとすると、底辺の長さはL＊COS(A)、残りの辺(角Aの向かい側にある辺)の長さはL＊SIN(A)となる。この性質を利用すると、おなじ長さの線分を自由な角度で表示できることになる。角Aのある頂点を基準点とすると、図1の斜辺とおなじ線分は、
LINE -STEP(L＊COS(A), -L＊SIN(A))
でかける（ＭＳＸの座標は上方向がマイナスになる)。角Aを変えていくと、線分の終点は円の軌動をえがいていくわけだ。
ところで、重要なのはMSXのBASICでは三角関数の引数の単位がラジアンになっているということだ。くわしくは114ページのこの欄に続く。

■図1 三角関数

ラインダンスのためのLINE文STEP指定

年末はいそがしい。あんまりいそがしいので、ちっともいそがしくなかった幼稚園のころがなつかしくなった。あのころはよかった。ひらがなばかりの絵本が読めるというだけで頭をなでなでしてもらえたし、先生のいうとおりに体を動かしているだけでダンスがおじょうずだといってもらえた。

お遊戯の時間に、みんなでお手々つないで首をかしげたりしながらおどった記憶があるが、あれはようするにラインダンスだったわけだなとおとなの感想が吹き出てきた。

ラインダンスといえば、いきなりだがＬＩＮＥ文だ。ふだんは点を座標で指定して線をかいたり箱をかいたりすることが多いけれど、もう1つ、STEPというオプションを使って、ダンスのステップをふむようにリズミカルな線をかいていくこともできる。

ふつうに座標を指定するやり方は「絶対指定」といって融通がきかないし、ちょっとしたことをやろうとしても座標の計算が面倒だが、STEPは下の書式で説明しているように「増分」(始点と終点の座標の差)を指定する「相対指定」なので、横方向、縦方向にいくつずつ進むかを考えるだけでいい。

とくに三角関数と組みあわせるとおなじ長さの線分をさまざまな角度でかくことができる。

始点を省略すれば線分を次々につないでかけるので、これを使って、正N角形を表示するサンプルプログラムを作ってみた。正N角形をかくための増分計算はいっけん複雑そうだが、三角関数以外は初歩の幾何で解決できるので、右ページの図もしばらくながめてみてほしい。いや、ただ見るだけじゃなくてね……。

●LINE文STEP指定の書式例

※同様に始点、または始点・終点の両方ともSTEP指定が使える。

```
LINE (x,y)-STEP(x',y')
```

始点(線のかきはじめの点)の座標を指定する。この部分を省略すると、そのまえにかかれた線の終点や、ほかのグラフィック関係の命令(DRAW、PSETなど)で指定された最後の点が自動的に始点になる。

座標のまえに「STEP」を入れると相対指定となる。x'、y'はそれぞれX、Y座標の増分。たとえばSTEP(x', y')をSTEP(10, 10)にすると、始点から右に10、下に10進んだ点までを線で結んでくれる。

正N角形をかく

正N角形の描画プログラム

```
10 COLOR 15,Ø,Ø:SCREEN 1:KEYOFF
20 PI=3.14159:R=1.11
30 INPUT"なんかっけいて゛すか";N
40 A=PI/2-PI/N
50 L=COS(A)*16Ø
60 DEFFNA=PI/2-A+(PI-2*A)*I
70 SCREEN 5:DRAW"BM12Ø,5"
80 FOR I=Ø TO N-1
90 LINE -STEP(INT(L*COS(FNA)+.5),INT(L*S
IN(FNA)*R+.5))
100 NEXT
110 IF INKEY$=CHR$(24) THEN 3Ø ELSE 11Ø
```

●外接円追加用プログラム

```
105 CIRCLE(12Ø,89+5),89,8,,,R
```

【プログラムの使い方】たとえば「なんかっけいですか？」で５を入力してリターンキーを押すと画面上部の点から１辺ずつ右回りに正五角形をかいていく。表示された図形を見あきたらSELECTキーではじめから。

【プログラムの説明】●行20の変数PIは円周率πのかわり。変数Rは歪み調整用。ディスプレイでは図形が少し横長になるため図形を縦方向に伸ばさないときちんと見えない。そこで行90でY方向の増分にRをかけて調節している。●行40～行60の計算式については左下の図とその上の説明参照。●行70のDRAW文は始点を座標(120,5)に置くためのもの(図形はなにもかかない)。●行80～行100で正N角形をかく。１辺ずつかくのでループ回数はN回。●行90のＬＩＮＥ文は左ページの書式とおなじ使い方。増分は(L＊COS(FNA), L＊SIN(FNA))でいいはずなのだが、このままでは小数点以下が単純に切り捨てられ誤差が大きくなる。INT(～＋.5)はこれを四捨五入して誤差を小さくするためのもの。

【追加プログラム】行105を加えると正Ｎ角形に外接する円を赤で表示するようになる。

※このプログラムはMSX2／2＋用ですが、行70のSCREEN5をSCREEN2に変えればMSX1でも動きます。

なんかっけいて゛すか？ 5

ここでたとえば5 を入力すると……

●正N角形の内角の2分の1にあたる角度Aは、三角形の内角の和はπ(180度)で、三角形ａｂｃが二等辺三角形であることから、A＝(π−2π／N)／2。整理すると行40の式になる。●N角形の1辺の長さLは、L／2がCOS(A)＊80になる(点ｂから直線ａｃに垂線をおろしたときにできる直角三角形から導く)ことから行50のように計算(80という数値はこの正N角形に外接する円の半径だが最終的には歪み率Rで約89になる)。●行60は１辺ずつかくときに折れ曲がるたびの角度を計算するためのユーザー関数を定義。この計算式の根拠は図２で解説。

正五角形を座標(120,5)から右まわりに１辺ずつかいていく

正七角形。これは分度器を使ってもかきにくい

36を入力すると円のような「正三十六角形」をかく

行105を追加すると正Ｎ角形に赤い外接円が加わる

このプログラムの正N角形に外接する円は半径80のはずだが、じっさいには歪み率1.11のために半径を80×1.11≒89にしてある。行105の最後のRは縦方向を変えずに横をR分の１にちぢめるもの。

■図２ 正N角形をかくための長さと角度の関係

ラジアン　角度の単位には２種類ある。１つはふだん慣れ親しんでいる「度」。この単位の場合は１周が360度になる。もう１つが、MSXのBASICや数学で使われている「ラジアン」。ラジアンの場合は、「°」のような記号をつけずに数値そのままで書くが、多くの場合、円周率πとのかけ算の形で現れてくる。たとえば、360度は２π、180度はπ、90度は$\pi/2$というふうに。ＭＳＸでは、90のかわりに$\pi/2$を使うようにしないと正しい計算にならないので、どこかでπを手にいれなければならない。

円周率π　πは3.1415926535……という無限に続く小数を持った定数で、そのうえ、MSXにはこのπをただでくれる機能がない。そこで、三角関数を正当に使うプログラムではπかπの倍数に近い値を持ったものをどこかで用意することが多いのだ。今回のプログラムで登場したPIやGがそれだ。たいていは、PAIとかPIとかPという変数に、そのプログラムに必要な精度に応じててきとうな位で四捨五入したπの近似値を設定している。113ページのプログラムでは小数点以下６桁で四捨五入したものを採用し（蛇足だが、もう１桁増やすと、「#」のマークがおしりについて取れなくなる。試してみよう）、115ページのほうでは小数点以下３桁を四捨五入したものを採用した。正直にいうと、正Ｎ角形のプログラムで採用したπの精度はたぶん不必要なものだろう。せいぜい3.1416くらいで十分だったろうが、なんとなく気分でもう１桁増やしてみただけなのだ。これも蛇足。

RGB　光の三原色（Red、Green、Blue）の頭文字を取った略語。つまり、アナログRGBディスプレイというのは、アナログ赤緑青ディスプレイのことだったわけだ。

SINとCOSの逆転　ここでとりあげなければ気がつかなかったかもしれないが、正Ｎ角形とアニメーションのプログラムでは大きくちがうところがある。前者ではＸ座標のほうにCOSを使い、Ｙ座標のほうにSINを使っているのに、後者ではそれが逆になっているのだ。アニメーションのプログラムのほうでもX座標にCOS、Y座標にSINを使ってもかまわないのだが、そうすると右端から時計まわりにまわりはじめることになる。ここでは、下端から反時計まわりにいくほうが自然な感じがしたのであえて逆にしてみた。一般に、SIN、COSを使って円運動などをやっているプログラムでは、SINとCOSを完全に入れ換えても基本的な動きは変わらない。ただ全体が90度向きを変え、進む方向が逆になったりするだけなのだ。

SINとCOSの関係　SINの２乗とCOSの２乗を加えると１になる。また、０度～90度ではSIN、COSともプラス、90度～180度ではSINはプラス、COSはマイナス、180度～240度ではSIN、COSともマイナス、240度～360度ではSINはマイナス、COSはプラスというふうに変化していくが、これはSINとCOSの値が、ちょうど90度ずれて揺れ動いているから。360度（２π）を超えるとどちらもまたもとにもどり、おなじことのくりかえし。

LINE文とパレットいじりでアニメーション

■正Ｎ角形プログラムの改造

正Ｎ角形のプログラムは、やはり「正」と名がつくものをやるだけに、画面の調整をしたり、誤差の調整をしたりで、雑音の多いプログラムになってしまったが、それさえなければ、基本は「おなじ長さの線」を「一定の角度ずつ曲げながら」「つないで」かくものだった。

この３つの要素のどれかをちょっと変えていろんなラインダンスをやってみると、ＭＳＸはスピログラフのかわりにもなるということがわかってくる。

たとえば、「だんだん短くなる線」を「少しずつ角度を大きくしながら」「つないで」かくようなプログラムは、正Ｎ角形のプログラムに、次のような改造を加えるとすぐにできる。

①行50の「160」を「200」に変える
②行60の最後に、「＋I（アイ）＊．5」をつけ加える
③行80の「N」を「L」に変える
④行90の２か所の「L」を「（L－I（アイ））」に変える

このとおりに修正すれば三角や四角に近い形が渦巻いていくようなグラフィックが表示され、それなりに美しい。

ほかにもてきとうに改造してみると、おもしろいことが起きるかもしれない。

■パレットでアニメーションも

１つのプログラムを改造ばかりしていてもあきてしまうので、もう１個作ってみた。こんどは、「おなじ長さの線を」を「一定の角度ずつ曲げながら」「始点を回転させつつ」かいてみよう。つまり、右ページの図４のように動いていくものだ。

↑右のプログラムの実行例の１つ。陽気な学校の校章のように見えないこともない

ついでに、新しい色鉛筆のセットを手に入れた幼稚園児になった気持ちで、１本の線をかくたびにCOLOR文で色を変えることにしてみた。すると、たとえば上のような図形になる。

さらについでに、カラーパレットを連続的に変化させてみるとこの模様がモヨモヨ動くようになる。この手法は、『グラフィックアニメーション』というプログラムに刺激されてファンダムで一時期流行したやり方だ。

たとえば、Ａ、Ｂ、Ｃと３つのカラーコードの線がならんでいて、Ａが黒、Ｂが灰色、Ｃが白だとする。次にＡが白、Ｂが黒、Ｃが灰色になり、その次にＡが灰色、Ｂが白、Ｃが黒になり……というふうに色の対応をずらしていくと動いているように見えるわけだ。ようするに電光掲示板とおなじ原理だ。

以上の要素をまとめたのが右ページのプログラム。パターンをかくのに１、２分かかるがまあいいじゃないか。

■図3　光の3原色

R
G
B

●パレット設定の書式

```
COLOR=(c,n,m,l)
```

c：色を変えたいカラーコード（MSX２以降では正式にはパレットコードという）を入れる。もとの色がどういう色かはとりあえず関係ない。ところで、パレット機能はSCREEN０～７のどのモードでも使えるが、SCREEN８でだけは使えない。

n～l：の順に、Ｒ（赤）、Ｇ（緑）、Ｂ（青）の色の明るさ（輝度）を０～７の数値で指定する。たとえばぜんぶ０にすると黒、ぜんぶ７にすると白。この組み合わせでできる色数は512色あることになる。

時間のかかるグラフィック遊び

BASICピクニック

パターンアニメーションのプログラム

```
10 SCREEN 1:COLOR 15,0,0:KEYOFF
20 INPUT"なんかいてん";K
30 INPUT"なんほ゛ん";B
40 INPUT"なか゛さ";L
50 SCREEN 5:G=6.28*K
60 FOR I=0 TO G STEP G/B
70 X=SIN(I)*L:Y=COS(I)*L
80 SX=128+SIN(I/K)*50:SY=96+COS(I/K)*50
90 C=(C-1)MOD13+2:COLOR C
100 LINE (SX,SY)-STEP(X,Y)
110 NEXT:FOR I=0 TO 1:I=-STRIG(0):NEXT
120 FOR I=1 TO 7 STEP 3
130 C=(C-1)MOD13+2
140 COLOR=(C,I,I,7)
150 NEXT:IF STICK(0)=1 THEN 10
160 GOTO 120
```

●また別のバリエーション追加用

```
70 X=L*3/5:Y=L*4/5
80 SX=20+200*I/G:SY=120+SIN(I)*L-40*I/G
```

なんかいてん? 6
なんぼん? 500
ながさ? 30

⬆まず3つの数値を設定

上の追加用リストを上のプログラムの行70、80と置き換えると、右下方向に固定された線分が波打つようなパターンをえがく(下段の写真)。したがって、「なんかいてん?」のときにする設定は回転ではなく、何回波打つかの指定に変わる。

【プログラムの使い方】「なんかいてん?」では、始点が1周するまでに線が何回転するかを指定。「なんぼん?」では、かきおわるまでに何本の線をかくかを指定。「ながさ?」では、かいていく線分の長さをドット数で指定。パターンをかきおわったらスペースキーを押すとアニメーション開始。あきたらカーソルキーの上を押すとまたはじめにもどる。

【プログラム解説】●このプログラムでは正N角形のプログラムとちがって、誤差や画面の歪みがさほど気にならないのでそのへんの調節はすべてはずした。●行50のGは、行60からはじまるループの最終値。Kは線分の回転数だが、そのまえの6.28はじつは2π(360度)のてきとうな近似値。つまり、1周ぶんの角度だ。●行60で「STEP G/B」としているのは、ループがはじまっておわるまでB本の線分をかくため。このままだとじつはB+1本の線分をかくことになるのだが、数えてみるとB本しか表示されない。じつは最初にかかれる線だけ黒なので見えないのだ。最初の線はかわいそうだが、こうしたほうがプログラムはシンプルになる。●行70は線分をかくための増分X、Yを計算。線分の長さと線の方向の角度から三角関数を使って計算。●行80では線分の始点の座標SX、SYを計算。三角関数の引数が「I/K」となっているのは、始点には1周しかさせたくないため。Iが最終的にGに達したとき、I/Kは6.28(360度)になっている。また、SX、SYに加えられている定数128と96は、始点がえがく円の中心座標。●行90はパレットコードを2〜14の順に次々に変えながら行100でかく線分の色を設定する。●行120〜150ではパレットコードを青系の3段階の色でずらしながら設定していく。ここがアニメーションの処理になる。

⬆6回転、500本、長さ30と指定してえがいたパターン

⬆スペースキーを押すとモノトーンになって動きはじめる

■図4 ラインの動き1

⬆別のバリエーション。2回転200本、長さ30で設定

⬆これも動く。線をかいているときの動きは右の図5参照

■図5 ラインの動き2

足したり比べたりできる文字の演算

BASICテクニック

#9　文字列のクッキング

BASICのプログラムのことを考えていると、ついつい数値に関わることばかりになってしまうけれど、たまには文字列の操作も見なおしてみよう。

文字列の演算　足し算ができるといっても、引き算や掛け算はない。足し算というよりは、むしろ文字列の「連結」といったほうがいいだろうか。ところで、文字列を表示するときに、この「＋」と「；」はおなじように働き、

PRINT　″A″＋″B″………①
PRINT　″A″；″B″………②
PRINT　″A″″B″…………③
PRINT　″AB″………………④

の4つともまったくおなじ表示になる。③は②のセミコロン（；）を省略したもの（文字変数でも同様に省略でき、じっさいによく省略される）。
ところが、長い文字列を使っている場合、Out of string space（文字列領域が不足している）というエラーは、②では出ないが①の形だと場合によっては出ることがある。①では「演算」をおこなっているが、文字列の演算は文字列領域を使っておこなわれるからだ。ちなみに②～④はまったく文字列領域を使用しない。

MML中で数値変数を使う　MML（PLAY文データ）は文字列だが、そのとちゅうで次のようにして数値変数の値を受け取ることができる。
″〈コマンド〉＝〈数値変数〉；″
〈コマンド〉は、O、L、T、VなどのMML中の文字だ。この形式にしておくと、数値変数の値がそのコマンドで指定する数のかわりになってくれる。
また、次のようにすれば文字変数をMML中に挿入することもできる。
″……X〈文字変数〉；……″
これもなかなか便利だ。この2つの使い方は、DRAW文のGML（グラフィックマクロランゲージ）でもおなじ。

BASICは文字に強い

BASIC以外のコンピュータ言語を知っているプログラマにいわせると、BASICはコンピュータ言語のなかでも文字に強い言語なんだそうだ。

そういうことなら、もう少し文字のことを意識してみたい。

■文字列の演算

たとえば、

A$＝″A″＋″B″

を実行するとA$には「AB」という文字列が入る。つまり、足し算ができるわけだ。

もう1つ、文字列の「大小」を次のような基準で判定する関係演算が使える。

①比較する文字列の頭から1文字ずつ取り出し、2つの文字が異なる場合にキャラクタコードで大小を判定

②比較のとちゅうで一方の文字列がなくなった（短かった）場合は、短いほうを小さいと判定

だから、たとえば、

″AB″＝″AB″
″AB″＞″AA″
″AB″＜″ABC″

といった関係が成り立つ。これは、いくつかの文字列グループをアルファベット順や50音順にならべるときなどに使える。

■応用範囲の広いMID$

文字列演算は以上の2種類しかないが、文字にまつわる関数は10数種類ある。なかでももっともよく使われる文字関数が、下図で紹介しているMID$だ。ある文字列中の何番目かの文字から何文字かを取り出す関数で、これは文字を順番に取り出して文字列をなにかのデータとして使ったり、すでにある文字列から新しい文字列をつくり出したりなど、応用範囲が広く、さまざまな場面で使われる。

●MID$関数の書式

```
MID$(〈文字列〉,n1,n2)
```

〈文字列〉：もとになる文字列を表す文字式（文字定数や文字変数も含む）。

n1：もとになる文字列の左から何番目の文字から取り出しはじめるかを指定。n1が0以下だとエラーになる。

n2：n1で指定した位置から取り出す文字数をn2で指定。取り出せる文字数より大きい数を指定してもエラーにはならない（ただし255以内）。また、省略すると指定位置から最後の文字までを取り出す。

たとえば文字列でこんなことができる

MID$でピアノのレッスン

```
10 A$="CDEFGABCDEFGA"
20 PN$="13654543"
30 PLAY"L16T120S0M6000"
40 FOR I=0 TO 7:FOR J=1 TO 8
50 N=VAL(MID$(PN$,J,1))+I
60 P$=MID$(A$,N,1):OV=4-(N>7)
70 PLAY "O=OV;"+P$
80 NEXT:NEXT
```

●近所にピアノを習っている人がいると、けだるい日曜の午後などに「ドミラソファソファミ」という、1音ずつ上がっていくメロディが聞こえてくる。このメロディパターンをプログラムにしてみた。●A$はドから1オクターブ上のラまでの白鍵。PN$は例のメロディパターンを数字で表したもの。●行30は音の長さ、テンポ、エンベロープの初期設定。●ループ変数のIは基準音を1音ずつ上げていくため。Jはメロディパターンを順に取り出すため。●Nは白鍵の位置、P$はその白鍵に対応する音名。OVは白鍵の位置に応じたオクターブ指定用。●行70の「=OV;」はMML中で数値変数を使うためのもの(左ページの傍注参照)で、OVの値に応じたオクターブを指定する。そのあとの「+P$」が行50、60で決定した音を演奏する。P$決定から演奏にいたる過程は下の図参照。

たとえばIが3、Jが4のとき

文字列 | 1 | 3 | 6 | 5 | 4 | 5 | 4 | 3 |

4　3

N=VAL(MID$(PN$,J,1))+I

PN$の4番目の文字から1文字取り出す──"5"

それを数値にする──5

5+3で8

P$=MID$(A$,N,1):OV=4-(N>7)

文字列 | C | D | E | F | G | A | B | C | D | E | F | G | A |

A$の8番目の文字から1文字取り出す──"C"

4-(-1)で5

PLAY "O=OV;"+P$ ──→ PLAY "O5C"

"O5"　"C"

行50でNにメロディパターンに応じた白鍵の位置が入る。行60ではそれに対応した音名をA$から取り出し、OVにもそれに応じた数を入れる。この場合はけっきょく"O5C"を演奏することになる。

10個の文字列を順序よくならべなおす

```
5 COLOR 15,4,7:SCREEN 1:WIDTH 29
10 FOR I=0 TO 9:INPUT A$(I):NEXT
20 FOR I=0 TO 8:FOR J=I+1 TO 9
30 IF A$(I)>A$(J) THEN SWAP A$(I),A$(J)
40 NEXT:NEXT
50 FOR I=0 TO 9:PRINT A$(I):NEXT
```

●ソフトハウスの名前をてきとうに入れてみると、さっと50音順にならべかえてくれる

練習問題

①〜④の空欄に2つの文字列の関係を表す演算子を入れなさい。

"A" ①□ "B"
"MSX" ②□ "FSX"
"FANDOM" ③□ "FUNDOM"
"MSX2" ④□ "MSX2+"

【解答】①<②>③<(最初にくいちがう文字によって大小を決定)④<(この場合は長いほうが大きい)

●左のプログラムをRUNすると入力待ちになるのでてきとうな文字列を10個入力する(もちろん1個ごとにリターンキーを押す)と、それをキャラクタコード順にならべかえてくれる。入力する文字をアルファベット大文字だけ、小文字だけ、またはカタカナだけにしておけば、アルファベット順、または50音順にならぶ(ひらがなだけの場合は、キャラクタコードのせいで濁音が清音よりも先になってしまうことがある)。●行5は画面モードをいちおう設定しているだけなので、じつはとくに必要がない。●行10で文字配列変数A$(n)に10個の文字列を入力。もっとたくさん入れたい場合は、これよりもまえの行で必要なだけの配列を宣言しておいて、ループ終了値を書き換えればいい。●行20〜行40で文字配列をならべかえている。最初の文字列と2番目の文字列、最初の文字列と3番目の文字列……というふうに比較していき、最初の文字列よりも小さい文字列(つまり先にならぶべき文字列)があれば入れ換える(SWAP文で2つの内容を交換している)。すると最終的にもっとも小さい文字列が先頭にならぶことになる。次に2番目の文字列と3番目の文字列……というふうに順次やっていくと配列A$(n)は文字列の小さいほうから順にならぶことになるわけだ。この作業をソートという。

**BIN$**　この仲間に、HEX$(ヘクサデシマルダラー：じっさいにはヘックスダラーと呼ぶことが多い)、OCT$(オクトダラー)がある。HEX$は、数値を16進数の文字列に変換し、OCT$も同様に8進数の文字列に変換する。

STR$　この関数とまったく逆の働きをする関数が、まえのページで出てきたVAL関数だ。たとえば、
VAL ("1234")
は、文字列としての「1234」ではなく、数値としての1234になる。この関数は、DATA文から16進データなどを読んできて、それを数値にもどすときによく使われている。たとえば、A$に「FF」という文字が読みこまれたとすると、
A=VAL ("&H"+A$)
によって、変数Aには255(16進表記では&HFF)が入る。このやり方は、もっと長い文字列を読みこんでMID$を使って分解し、上の方法で数値化するというパターンで多用される。

INSTR　本文中の実例でYのまえに空白があるのは、この関数の困った性質のための対策だ。なにもキー入力されていないときは、INSTR(〈文字列〉, INKEY$)の値はかならず1になってしまうのだ(どんな文字列でもそうなる)。だから、意味のある判断をするためには、1を無視する必要があるので、本文中の例のように1文字目に無視できる文字を置くことが多い。もっとも、INSTRのまえにキー入力があるかないかを判断すれば問題はないのだが。それにしても、この現象はなんだか『不思議な国のアリス』みたいでおもしろい。あらゆる文字列の最初には「文字がない」がある、ということなのだろう。また、いいわすれたが、INSTRが文字を調べるとき、1つ見つかればそのあとにおなじ文字があっても無視される。

STRING$　この仲間に、
SPACE$(n)
というのがある。これはn個ぶんの空白を連結した文字列を生み出す関数だ。

表示を整える文字関数　文字関数のなかでも変わったタイプの関数に、
SPC (n) ……………………①
TAB (n) ……………………②
の2つがある。これはどちらもPRINT文中でしか使えない特殊なものだ。①は、n個の空白をPRINTし、②はそのときの位置から空白をPRINTしながらX座標nの位置にカーソルを移動させる働きがある(最初の位置がX座標nより右側になるときは無視される)。

CLEAR文　文字列の演算が多いプログラムではメモリ内の文字列領域を大量に使用する場合があるので、通常の状態でやるとOut of string spaceというエラーが出ることがある。そういうときは、次のようにCLEAR文を使って文字列領域(初期値は200バイト)をあるていど広げておく必要がある。
CLEAR 〈数値〉
の〈数値〉のところの数値を変えるのだ。ただ、あまり大きく取りすぎるとこんどはOut of memoryが出ることがあるので、エラーが出ないていどの大きさ+アルファくらいにしておいたほうが無難。

そのほかの文字にまつわる関数

■RIGHT$(ライトダラー)とLEFT$(レフトダラー)

文字列からある文字を取り出す関数には、ほかにRIGHT$、LEFT$がある。これは文字どおり、右または左から何文字かを取り出す関数だ。
RIGHT$(〈文字式〉, 〈取り出す文字数〉)
のように使う。

右ページのサンプルでは、トランプのシャッフルにこの2つの関数を使ってみた。トランプの山の上と下をLEFT$とRIGHT$で表現したのだ。

■2進数変換のBIN$(バイナリーダラー)

たとえば、ある数を2進数8桁にそろえて表示したい場合、
PRINT RIGHT$("Ø0000000"+BIN$(〈ある数〉), 8)
といった使い方をする。

BIN$は、文字の型を変換する関数で、ある数を2進数の文字列に変換する。たとえば、
9→1001
23→10111
というふうに。しかし、このままでは数によって桁数のちがう2進数しか出してくれない。そこで、BIN$の結果の頭に「Ø」を7つくっつけて、その結果の文字列の右から8桁取り出しているのだ。こうしておけば、
9→00001001
23→00010111
というふうに桁がそろう。これを使ってVRAMに入っている文字やスプライトのパターンを読み出して表示すれば、Øと1でできたパターンを画面で確認することができるわけだ。

■数値を数字に変えるSTR$(ストリングダラー)

STR$関数(数値を数字に変える)の補正用にRIGHT$を使うこともよくある。

STR$関数は、正の数を数字に変えるときに数字のまえに空白を1つくっつけてしまうのだ。このためにSTR$で変換した数字はそのままほかの文字列とくっつけたりするわけにはいかないことが多い。

そこで、この空白を取り去るためにRIGHT$と、LEN(レングス)関数(文字の長さを調べる)が登場する。
A$=STR$(〈数値〉)
とやったうえで、次の2つの結果を比べてみると効果がはっきりわかるだろう。
PRINT A$
PRINT RIGHT$(A$, LEN (A$)−1)
RIGHT$とLENを使った後者は、純粋に数字だけになっている。

■文字列を検索するINSTR(インストリング)

LEN関数はほかの文字列を作る関数とはちがって文字列を調べるものだが、もうひとつたいせつな関数にINSTR関数がある。

基本的には、ある文字列(これをA$とする)が別の文字列(これをB$とする)のどこにあるか(またはないか)を調べる関数で、
INSTR (〈調べはじめる場所〉, B$, A$)
という書式で使う。調べはじめる場所とは、B$の何文字目から調べていくかを指定するもの。この部分はコンマごと省略できて、
INSTR (B$, A$)
という形で使われることが多い。省略した場合は、文字列の頭から調べていくことになる。

この関数で調べた結果、A$がB$のn文字目にあったとすると、nという数値を返してくる。また、B$にA$が含まれていなかったときは0になる。

じっさいによく見られるのは、キー入力のときにどの文字を押したかを調べるケースで、
INSTR(" YyNn", INKEY$)※文字列の頭に空白があることに注意(傍注参照)
という形になることが多い。

■まとめて作るSTRING$(ストリングダラー)

CHR$関数やASC関数も文字関数の一種だが、CHR$関数の機能を拡張したような関数にSTRING$関数がある。これはSTR$関数と読み方がおなじで混同されやすい。

STRING$関数は、
STRING$(〈作りたい個数〉, 〈作りたい文字のキャラクタコード〉)
という書式で使われ、指定した文字を指定した数だけつないだ文字列を作り出す。キャラクタコードのところには数値のかわりに文字式も入れられる(ただし、その最初の文字だけがこの関数の対象となる)。

ほかにも、いくつかの表示を整える文字関数がある。

これらの関数を使って文字の操作や表示をあれこれ考えているといろんなことができそうだ。

⇧右のプログラムを実行。ばふばふシャッフルとしゅわわシャフルを1回ずつやったところ

トランプゲームへの入口

2とおりの方法でトランプをシャッフルする

```
10 COLOR 15,1,1:SCREEN 1:WIDTH 26
20 CLEAR 1000:CARD=RND(-TIME)
30 N$="A23456789^JQK":S$="♠♣sq"
40 FOR I=0 TO 23:VPOKE 904+I,VPEEK(1032+
I):NEXT:FOR I=0 TO 6:VPOKE 752+I,VAL("&H
"+MID$("4CD252525252524C",2*I+1,2)):NEXT:V
POKE 8206,&H8E:VPOKE 8208,&H1E:FOR I=0 T
O 5:VPOKE 8198+I,&H4E:NEXT
50 FOR I=1 TO 4:FOR J=1 TO 13:A$=A$+MID$
(S$,I,1)+MID$(N$,J,1):NEXT:NEXT:B$=A$
60 PRINT "HAJIME":PRINT A$
70 FOR K=1 TO 3
80 FOR I=0 TO 10:T=INT((RND(1)*10+10))*2
+1:M=INT((RND(1)*30+20))*2:B$=MID$(B$,T,
M)+LEFT$(B$,T-1)+RIGHT$(B$,(T+M-105)*(T+
M<106)):NEXT
90 PRINT "BAFUBAFU";K:PRINT B$
100 C$="":FOR I=0 TO 25:C$=C$+MID$(B$,I*
2+1,2)+MID$(B$,I*2+53,2):NEXT:B$=C$
110 PRINT "SHUWAWA";K:PRINT B$
120 IF STRIG(0) THEN NEXT ELSE 120
```

●トランプゲームを作るためにはトランプ1組とそれをシャッフルする方法が必要だ。トランプ1組を文字で作るのはわりとかんたんだが、問題はシャッフル。じつは1～52の重複しないランダムな数列を作っていけばわりとかんたんなのだが、それではなんとなく味気ないような気がするので、ここではスペードのAからハートのKまでが整然とならんだトランプの山(文字列)を2とおりの方法でじっさいにシャッフルするプログラムを作ってみた。プログラムを実行すると、まず「HAJIME」のあとに最初のトランプの山を表示し、「BAFUBAFU」のあとに、ばふばふシャッフルしたトランプを表示、さらにそれをしゅわわシャッフルしたトランプを「SHUWAWA」のあとに表示して止まる。スペースキーを押すと、2回目のシャッフルをして同様に表示し、3回目のシャッフルで終了する。●行20のCLEAR文は文字スペースを1000バイト取るため。文字列の演算のためにメモリの文字スペースをくうのでちょっと多めに設定しておいたのだ。そのあとのCARDという変数にはとくに意味はなく、乱数系列の初期化をしているだけ。●行30、40ではトランプ1組用文字列の素材とパターン定義、カラー定義をおこなっている。「sq」は赤い「◇♡」に変わる。●行50でトランプ1組ぶんの文字列を作り、A$に入れる。シャッフルのためにB$に入れ直している。●行80はばふばふシャッフル。変数Tは、ばふっとカードを抜くときの取り出しはじめる位置(上から何枚目か)、Mは取り出す枚数。上からT番目からM枚取り、それを上に重ねるのが、ここでいうばふばふシャッフルだ。●行100は、しゅわわシャッフル。これはかんたんで、山を2つに分け、1枚ずつ交互にしゅわわっと重ねていくだけ。

●ばふばふっと切る

52枚ものトランプを図にするとたいへんなので、20枚だけでばふばふシャッフルの仕組みを図示した。T番目のカードからM枚取り出し、残りを重ね、そのうえに取り出したカードをもどす。

ばふばふシャッフルの図

●しゅわわっと切る

じっさいのトランプではこちらのほうがむずかしいが、プログラムではかんたんだ。トランプ1組の文字列の上半分と下半分から交互に1枚ずつ取り出して重ねていく。

しゅわわシャッフルの図

DRAW文とGMLと愛について

BASICテクニック

#10 DRAW文で愛の告白

チョコレートのおいしい季節、ふだんあまり見かけないDRAW文に注目してみよう。DRAW文のなかにはもうひとつの「言語」が隠されているのだ。

WIN文とLOSE文　もちろん、こんなものはもともとない。念のため。
GML　Graphic Macro Language(グラフィック・マクロ・ランゲージ)の略で、グラフィックマクロ命令とも呼ばれる。ようするにDRAW文のグラフィックデータのこと。33ページのGML一覧表参照。GMLは、おもにアルファベット1文字の「コマンド」とそれに付随する数の組み合わせでできている一種の命令体系、すなわち「言語」で、文字列として扱われる。文字列であるからには、文字列演算を組み合わせて、グラフィックデータをさまざまに加工することができる。この点は、PLAY文のMMLによく似ている(間接指定モードの書式などはまったくおなじ)。
LP　Last access Point(ラスト・アクセス・ポイント)の略。最終参照点とか、たんに参照点とも呼ばれる。おおざっぱにいうと、DRAW文やLINE文などのグラフィック命令でなにかをかくときのペン先の位置のようなもの。たとえば、LINE文で点Aから点Bまで線を引くと、LPは点Bの位置に移動する。このときにDRAW文で座標指定なしに線をかくと、始点はこの点Bになるわけだ。ただし、CIRCLE文を実行したときのLPは、円の中心になっているので注意。PAINT、PSET、PRESET、PUTSPRITEで指定された座標もLPとなる(スプライト自体はグラフィック座標上では1ドット下に表示されるので注意)。また、グラフィック画面上に文字を表示したときも文字量に応じて移動する。

文字列でパターンを描くDRAW文

むかし、BASICには、光で輝く線をかくWIN(ウイン)文と、闇に隠された線をかくLOSE(ルーズ)文の2種類があった。正反対の2つの文は、たがいに憎みあい、泥沼の戦いをくりひろげていたが、やがて決着のつくときがきた。引き分けたのである。そして、DRAW(ドロー)文が生まれた。

DRAW文は、グラフィックデータを文字列の形であつかう。

文字列といってもただの文字列ではない。GMLと呼ばれる一種の言語なのだ。慣れてくるとこれがけっこうおもしろい。

33ページにGMLに含まれるコマンドの一覧表を掲載してあるので、いまはじめてGMLの存在を知った人はまずそちらから見るといい。

季節がら、「あいしてる♡」というメッセージをDRAW文でかきつつ、DRAW文の使い道について考えてみよう。

↑リスト1の実行画面。書き順どおりに、「あ」の横棒からかきはじめる

↑そして縦棒。鉛筆の形をしたスプライトはかかれる線といっしょに動いていく

↑「あ」が完成した。文字の形はMSXのひらがなを参考にして作った

↑ハートマークまでをDRAW文でかいて最後にハートのなかを赤くぬりつぶす

↑赤いハート付き愛のメッセージの完成。Sコマンドで大きさも変えられる

バレンタインデーのお返しに

リスト1：愛してるパート1

```
10 COLOR 1,14,14:SCREEN 5,1
20 SPRITE$(0)="みネしセ"+CHR$(127)+">"+CHR$(
28)+CHR$(8)
30 DRAW"S16A0BM20,100C1"
40 FOR I=0 TO 5:READ A$:G=LEN(A$)
50 FOR J=1 TO G STEP 3
60 DRAW MID$(A$,J,3)
70 PUTSPRITE0,STEP(0,0),1:FOR W=0 TO 40:
NEXT
80 NEXT J,I
90 DRAW"BM-5,3":PAINT STEP(0,0),8,1
100 FOR I=0 TO 1:I=-STRIG(0):NEXT:RUN
110 DATA BD1R6 BL4BU1D6 BE3G3 LHUER4FDGB
R3BU6:'あ
120 DATA BD1D4 FREBE3D2 BR2BU4:'い
130 DATA D5FR2 E2 BR2BU4:'し
140 DATA BD1R6 GLGLGDFR2BR4BU6:'て
150 DATA BR1R4 G2 LG2ER4FD GL4HE RFGBR6B
U6:'る
160 DATA BD3BF3H3 U2ERFERFDDG3BR5BU6:'♥
```

●リスト1は、鉛筆の形をしたスプライトが画面を左から右へシュシュッと動きながら、「あいしてる♡」というメッセージをかくプログラムだ。最後にPAINT文でハートを赤くぬって止まる。スペースキーを押せばまた最初からかきはじめる。●行20は、鉛筆スプライトのパターン定義。●行30は、DRAW文の初期設定。倍率を16（1移動単位あたり4ドット）、回転を0（初期状態）、書きはじめの位置を座標（20,100）、線の色を黒に設定している（各コマンドの意味は33ページ参照）。倍率や回転など、まえに走ったプログラムでの設定が残るのでこうした初期設定は入れたほうが安全。●A$に各文字用のGMLが読みこまれる（Iのループ1回ごとにメッセージ1文字ぶん）。GMLの長さを変数Gに入れるのはJのループの回数を決めるため。●行50からのJのループが1文字ごとの描画ルーチン。行60で読みこんだGMLを3文字ずつ次々と引き出して線をかいていく。●行70では、スプライトの相対座標指定を使って、LP（おもに最後に図形をかいたときの終点の座標。GMLの場合は、まえのコマンドによる図形の終点で、次にかく図形の始点の座標でもある。30ページの傍注参照）にスプライトを表示している。●行90はハートの真ん中あたりにLPを移動し、ハートの中を赤くぬりつぶしている。●行110からが各文字用のGML。基本的には下図のように8×8のマスの左上から開始し、字を書き終わると次のブロックの左上に行くようなデータにしてある。

DRAW文が「あ」をかいていく全プロセス

行110の「あ」のGMLがたどる軌跡を図解した。黒矢印ではLPのみが移動し、赤矢印で線がかかれている。最後に矢印が枠からはみ出ているのは次の文字（「い」）の左上にLPを移動させるため。1マスは1移動単位を表す（このプログラムの場合は1移動単位が4ドットになっている）。

①BD1……線をかかずに下へLPを1マス移動する。

②R6……右へ6マス移動した地点まで線をかく。

③BL4……線をかかずに左へLPを4マス移動する。

④BU1……線をかかずに上へLPを1マス移動する。

⑤D6……下へ6マス移動した地点まで線をかく。

⑥BE3……線をかかずに右上へLPを3マス移動する。

⑦G3……左下へ3マス移動した地点まで線をかく。

⑧L……左へ1マス移動した地点まで線をかく。

⑨H……左上へ1マス移動した地点まで線をかく。

⑩U……上へ1マス移動した地点まで線をかく。

⑪E……右上へ1マス移動した地点まで線をかく。

⑫R4……右へ4マス移動した地点まで線をかく。

⑬F……右下へ1マス移動した地点まで線をかく。

⑭D……下へ1マス移動した地点まで線をかく。

⑮G……左下へ1マス移動した地点まで線をかく。

⑯BR3……線をかかずに右へLPを3マス移動する。

⑰BU6……線をかかずに上へLPを6マス移動する。

※この記事に掲載しているプログラムは、すべてMSX2以上の機種用ですが、リスト1、リスト2はSCREEN5をSCREEN2に変更し、行90を削除すればMSX1でも動きます。また、「おまけ」は、SCREEN文の変更だけでMSX1用になります。

GMLの暗記法　GMLコマンドはアルファベット１文字なのではじめは覚えにくいが、それぞれある英単語の頭文字から来ているようだ。以下はすべて推測だが記憶の補助にはなるだろう。
●方向・距離指定の移動コマンド
U＝Up(上)
D＝Down(下)
R＝Right(右)
L＝Left(左)
斜めのコマンドは、右上方向からぐるっと右まわりにEFGHとアルファベット順になっているので、１つ１つはピンとこなくても図をかけばすぐに思い出せる。ところで、なぜ、Ｅからはじまっているのだろうか。ほんとうはＡから始めたかったところだろうが、Ａ～Ｄはほかのコマンドに使われているためだと思われる。
●座標指定の移動コマンド
M＝Move(移動する)
●修飾コマンド
B＝Blind(盲目の、隠れた)※線をかかないから
N＝Not(否定の副詞)※LPを移動させないから
●特殊コマンド
S＝Scale(規模)
A＝Angle(角度)
C＝Color(色)
倍率　Ｓコマンドで指定される数値の４分の１がじっさいの倍率(スケールファクタ)になる。この「４分の１」というのがクセモノだ。
Ｓコマンドで指定される数値をｓ、移動コマンドで指定される数値をｎとすると、移動コマンドによる、じっさいの移動ドット数は、
ｎ×ｓ÷４ドット
になるので、たとえば「Ｓ１ØU３」とした場合、移動ドット数は、
３×10÷４＝7.5ドット
小数点以下は切り捨てられるのでじっさいには７ドットになる。まあ、いいけどね、なんていってはいられないのだ。なぜなら、このために、「Ｓ１ØU３U３」と「Ｓ１ØU６」の２つのGMLの実行結果がくいちがってしまうからだ。前者は７＋７で14ドットしか移動しないのに、後者は15ドット移動するのだから(これは物をバラバラに買うより、まとめて買ったほうが何円か消費税を節約できるのとおなじ理屈)。このため、この記事で使用するプログラムは、このクセモノとの直接対決を避けて、すべてＳコマンドでは４の倍数(16)を指定している。しかし、試しにリスト１や２の「Ｓ１６」を「Ｓ１Ø」にして実行してみたところ、「へたくそな字」になってけっこう味わいがあったので、それはそれなりにおもしろい面もあるわけだ。

DRAW文メッセージの2つの道

あたりまえだが、ＰＬＡＹ文で音楽を演奏する場合、なによりも演奏データとなるMMLの組み立てが重要なように、ＤＲＡＷ文でも描画データとなるＧＭＬの組み立てがいちばんのポイントになる。

この記事で使った文字の形は基本的にＭＳＸのふつうの文字パターンをもとにした(ただし、「し」は文字の右側にあいている空白をすこしせばめ、「る」は文字の形をちょっと変えている)。リスト１の行30、リスト２の行40にあるＳコマンドを「Ｓ１６」から「Ｓ４」(１移動単位が１ドットになる)に変更して実行してみると、ＳＣＲＥＥＮ１で表示される文字とほぼおなじ形で表示される。

どの文字用のGMLでもはじまりと終わりにＢコマンド(線をかかずにＬＰだけを移動する)を使った部分があるが、これは、どの文字をかく場合もかならず左上からはじまって、最終的に次の文字の左上までＬＰが移動するようにしたかったためだ。こういう部分を付けておかないと、それぞれの文字を表示する位置がバラバラになってしまう。

また、リスト１、２とも、３文字単位でGMLを取り出すため、かならず３文字単位で意味のあるGMLになるように作ってある。GMLのところどころにスペースが入っているのは、そのせいだ。

■リスト１とリスト２のちがい

ところで、リスト１で使っているデータは、31ページの図に示したような手順をGMLにした単純なものだが、右ページのリスト２ではそのGMLを逆転して使っている。実行する順番も逆だし、動く方向も逆なのだ。

たとえば、リスト２の「あ」のGML(行160)を逆から取り出してみると、BU１、L６、BR４、BD１……となっているが、これはリスト１の「あ」のGML、BD１、R６、BL４、BU１……をそれぞれ正反対の方向に変えたものなのだ。

このように、GMLの順序と方向を逆にすると、おなじ形を逆方向にかいていくようになる。

リスト２は、逆方向にかくGMLを、メッセージとは逆の順序で連結しておいて(M$に入る)、それをさらにうしろのほうから取り出して実行している。

うしろから取り出す文字数を少しずつ増やしていくことで、１つの点から少しずつメッセージがわいてくる効果を出しているのだ。

このリスト２的手法を使って、ちょっとしたグラフィックデモのおまけも作ってみた(下のリスト)。

ほかにも、いろいろな文字列演算を組みあわせれば、GMLの可能性はいろいろありそうで、なかなか楽しい。

↑リスト２の実行画面。こんどは動かない鉛筆スプライトの先から文字が出てくる

↑鉛筆を固定し、そのかわり紙を動かして文字をかく感じ

↑「あ」の終わりのほうではぐるぐるっと文字全体をまわしながらかいていく

↑メッセージ全体を動かしながらかいていくので最後のほうではとても重たい感じ

↑最終的に表示されるパターンはリスト１とおなじ

```
10 COLOR 15,1,1:SCREEN 5
20 A$="UUEUERERRFDFDDGDGLGDGDDFDFRFRR"
30 DRAW"S8A0":S=LEN(A$)+1:E=1:Z=-1
40 FORI= S TO E STEP Z
50 P$="BM100,100"+MID$(A$,I)
60 DRAW "C0XK$;C15XP$;":K$=P$:NEXT
70 SWAP S,E:Z=-Z:A=A+.5+(A>3)*4
80 DRAW"A=A;":GOTO 40
```

おまけ：ちょっと不気味なちぢれっ毛

●ちぢれっ毛のようなパターン(じつはいいかげんに作ったGML)を画面中央の１点から出したり入れたりするデモのプログラム。１回ごとに方向を90度ずつ変えていく。●行20でちぢれっ毛パターンをA$に設定。このA$のうしろから文字を取り出していく。最初は取り出しはじめる場所をA$の外からはじめ(このときは文字が取り出せないのでなにも表示しない)、１文字ぶんずつまえのほうにずらして取り出す文字数を増やしながらパターンをかいていく(ちぢれっ毛が現れていく)。この部分はリスト２の解説も参照してほしい。すべて表示しおわったらこんどは逆の手順でもとにもどっていく(ちぢれっ毛がすいこまれていく)。変数S、E、Zは２つの操作を切り換えるための変数だ。●行70、80に出てくるAは、ちぢれっ毛を１回出し入れするたびに90度回転するための変数。

おしりから愛をこめて

リスト2:愛してるパート2

```
5 CLEAR 1ØØØ
1Ø COLOR 1,14,14:SCREEN 5,1
2Ø SPRITE$(Ø)="みネしセ"+CHR$(127)+">"+CHR$(
28)+CHR$(8)
3Ø FOR I=Ø TO 5:READ A$:M$=M$+A$:NEXT
4Ø DRAW"S16AØ":G=LEN(M$)-2:X=208:Y=1ØØ
5Ø PUTSPRITE Ø,(X,Y),1
6Ø FOR I=G TO 1 STEP -3
7Ø P$="BM=X;,=Y;"+MID$(M$,I)
8Ø DRAW"CØXK$;C1XP$;":K$=P$:NEXT
9Ø DRAW"BM=X;,=Y;BM-5,3":PAINT STEP(Ø,Ø)
,8,1
1ØØ FOR I=Ø TO 1:I=-STRIG(Ø):NEXT:RUN
11Ø DATA BD6BL5E3UUHLGHLGD2F3 BH3BU3:'♥
12Ø DATA BD6BL6EHLGF R4EUH L4GE2RE2 L4 B
L1:'る
13Ø DATA BD6BL4L2HUERERERL6 BU1:'て
14Ø DATA BD4BL2G2 L2 HU5:'し
15Ø DATA BD4BL2U2 BG3GLHU4 BU1:'い
16Ø DATA BD6BL3EUHL4GDFRE3 BG3U6 BD1BR4L
6 BU1:'あ
```

●鉛筆スプライトを右端に固定しておいて、そこから文字がわき出すように表示していくプログラムだ。かきおわったあとスペースキーでまたはじめからくりかえす。●行5のCLEAR文は、文字列領域を1000バイトに増やすためのもの(通常は200バイト)。このプログラムでは変数M$にメッセージすべてのGMLが入ることになり、そのための演算(行30、行70)のために大きな文字列領域が必要になっている。●行40で設定されている変数Gは、行60からはじまるループの初期値。M$の長さから2を引いた値になっているのは、最初はM$の末尾3文字ぶんのGMLからDRAW文で実行していくことになるからだ。●リスト1ではパターンをかきはじめる最初の座標をDRAW文の初期設定でおこなっていたが、リスト2では、スプライトの表示(行50)がそのかわりになっている。グラフィックのLPは、スプライトが表示されたときも移動するのだ。行40で変数X、Yはその座標値になる。●行70で毎回のDRAW文用GML(変数P$に入る)を作成する。リスト1とちがって、このプログラムでは毎回のLPがおなじ座標でなければ困るので頭に座標指定のGMLを入れている(X、Yの値は行40以来変化しない)。●行80がじっさいにパターンをかいている部分。変数K$は1回まえのパターンで、カラーコード0(透明)で表示することにより、まえにかいたパターンを消しているのだ。このため、少しチラチラする。●行90、100はリスト1と同様。●行110からのデータは、リスト1とは順番と方向が正反対になっている。その理由は左の本文中で説明している。

GML(グラフィックマクロランゲージ)一覧表

コマンド	書式	コマンドの意味と使用例
移動コマンド(方向・距離指定)		
U D R L E F G H	Un 上にn単位移動 Dn 下にn単位移動 Rn 右にn単位移動 Ln 左にn単位移動 En 右斜め上にn単位移動 Fn 右斜め下にn単位移動 Gn 左斜め下にn単位移動 Hn 左斜め上にn単位移動	●指定された方向(8方向)へ線をかいていくコマンド。方向は下図参照。それぞれのコマンドに続く移動単位の指定(各コマンドの直後につける数値。左の書式ではnで表記)を省略するとn=1と見なされる。移動単位の大きさは、右の段のSコマンドによって変わるが、初期状態(S4)では、1単位=1ドットになっていて、たとえば、《DRAW "U4E4R4F4D4G4L4H4"》とすれば、上に4ドット移動しながら線をかき、そこから右上へ縦横4ドットぶん移動しながら線をかき……というふうにして八角形を描くことになる。ただし、始点の1ドットは数えないので、nドット移動しながら線をかくと、最終的にn+1ドットぶんの線をかくことになるという点に注意しておこう。
移動コマンド(座標指定)		
M	Mx, y 座標(x, y)まで線をかく M±x, y X方向にX単位、Y方向にY単位移動して線をかく	●Mの直後に指定される座標まで線をかく。たとえば、《DRAW "M1ØØ,1ØØ"》ならLPから座標(100,100)まで線をかく。座標指定なのでSコマンドの影響はない。 ●Mの直後に+か-の符号があるとX軸方向にx単位、Y軸方向にy単位移動する相対指定モードになる。《DRAW "M+1Ø, 1Ø"》ならLPからX方向に10単位、Y方向に10単位移動する。この場合はSコマンドが影響する。

コマンド	書式	コマンドの意味と使用例
修飾コマンド(移動方法の指定)		
B	B〈移動コマンド〉 移動するが描画しない	●Bコマンドのついた移動コマンドは、LPは移動させるが、線をかかない。《DRAW "BM1ØØ, 1ØØ"》なら座標(100,100)にLPが移動する。座標指定によく使われる。
N	N〈移動コマンド〉 移動するがLPはもとのまま	●Nコマンドのついた移動コマンドは、線はかくが、LPを移動させない。《DRAW "NLNUNR"》なら1点を中心に左、上、右に線をかき、LPはもとのまま。
特殊コマンド(倍率、回転、色指定)		
S	Sn 1単位をn/4ドットに指定	●移動コマンドで指定する移動単位のドット数を指定する。Sの直後に置かれる数(書式ではnで表記)の4分の1が、1単位のドット数となる。《DRAW "S8U2"》なら上方向に4ドット移動する。初期状態ではS4になっている。
A	An n×90度左に回転	●あるGMLで描かれる図形を90度単位で左回りに回転して表示する。《DRAW "A1U2"》なら左に2単位移動。
C	Cn カラーコードnで描画	●DRAW文でかく線の色をカラーコード(パレットコード)で指定する。《DRAW "C1"》なら黒を指定。
間接指定モード(GML中で変数を使う)		
X;	X〈文字変数〉; 文字変数の内容をGMLとして描画	●文字変数をGMLの一部として組みこむ。前後にほかのGMLがあってよい。たとえば《A$="URDL"》としてあれば《DRAW "S16XA$"》は《DRAW "S16URDL"》とおなじで縦横5ドットの四角をかく。
=;	=〈数値変数〉; 数値変数の内容をGML中の数値とする	●数値変数をGMLの一部として組みこむ。たとえば《A=16》としてあれば《DRAW "S=A;URDL"》は《DRAW "S16URDL"》とおなじことになる。

電源を切ってマニュアルを閉じて記憶と推理のなかへ

BASICテクニック

＃11 机上のMSX

春になったことだし、4月号だし、ほんのお遊びでMSXに関するクイズを作ってみた。MSXを片づけて、紙とえんぴつと消しゴムだけでやってみよう。

CTRL＋A コントロールコードの1つで、キャラクタコードは1。コントロールコードとは、キャラクタコード0～31の特殊な「キャラクタ」(文字ではなく、電気信号＝符号としてのキャラクタ)で、画面を消去したり(CTRL＋L)、カーソル以降の文字を消去したり(CTRL＋E)、ビープ音を鳴らしたり(CTRL＋G)、いろいろなことをしてくれる。そのなかでもCTRL＋Aは変わり種で、グラフィックヘッダという符号を入力するものだ。グラフィックヘッダは、「月」などのグラフィックキャラクタを表示するときに、頭につけるキャラクタで、たとえば「月」をCHR$を使って表示するときは、

PRINT CHR$(1)+CHR$(65)

という形で実行しなくてはいけない。CHR$(65)そのものは「A」という文字だが(これはCTRL＋AのAとは関係ない)、そのまえにCHR$(1)を入れることで、「月」という文字に変えてしまうのだ。

SHIFTキーを押しながらかなキー MSX2以降の機種にある、ローマ字かな入力モードに入るための操作。このとき、CAPSがオンになっていればカタカナ、オフになっていればひらがなになるが、さらに文字が確定するときにSHIFTを押していると、その関係が逆転するのだ。

3つのキーを同時に押す MSXのハードの性質上、3つ以上のキーを同時に押すと押していないキーが押されたことになってしまう場合がある。問題3の結果は、そのためだ。

まぶたのMSXにきいてみよう

たとえば、知らなかった人にはわりとショックなキー入力のクイズから話をはじめよう。MSXで試したりせずに、頭のなかだけで考えてほしい。

・第1問

アルファベット大文字モードになっているとき、次の①、②の順にキーを押すと、どんな文字が表示されるか。

①CTRLキーを押しながらAキーを押す

②Aキーだけを押す

・第2問

おなじくアルファベットの大文字モードになっているとき、次の①～③の順にキーを押すと、どんな文字が表示されるか(MSX2／2＋のみ)。

①SHIFTキーを押しながらかなキーを押す

②Sキーだけを押す

③SHIFTキーを押しながらAキーを押す

・第3問

次の3つのキーを同時に押すと、どんなことが起きるか。

CTRLキー、SHIFTキー、スペースキー

答えは、このページの最後に書いてある。

こういうクイズは、MSXに電源が入っていればすぐにわかるわけだが、そういう「実験」をせずに頭のなかだけで答えを探ってみると正解にいたらなくてもなにかが見えるのではないかと思う。

そのなにかとは、MSXを擬人化していえば、MSXの「精神構造」のようなものだ。

MSXの精神構造は、人間のそれと本質的にちがう。人間は、あえていえば神が作った知的システムだが、コンピュータは古いことばでいえば「人工知能」のシステムだ。その「人工」のぶんだけ、不自然なところ、むだな部分がときどきある。

問題1～3の現象は、その1例だ。役にたつかたたないかはべつにして、少なくとも設計した段階ではそういう「機能」は計算に入れていなかっただろうと思われる。なぜなら、具体的にそのことが書いてあるマニュアルは存在しないからだ。

こういう不思議な現象などを含めて、MSXに関するクイズを作ってみた。なぜいまクイズなのかと問いつめられると、なんとなく春だからとしかいいようがないが、最初はクロスワードを渋茶でもすすりながら解いてウォーミングアップしてじっくりとりくんでみてほしい。

では、問題1～3の答えを書く。あとで、MSXの電源を入れて試してみよう。

問題1→「月」
問題2→「さ」(ひらがな)
問題3→CLSが起こる

MSX非公式模擬試験問題A

問1

【クロスワードパズル】X軸(横)方向のカギとY軸(縦)方向のカギにふさわしいことばで下図の空欄を埋め、クロスワードパズルを完成しなさい。1マスにアルファベット1文字が入る。また、登場することばは、MSX、またはBASICに関するものだけで構成されている。

1 I	2 F	■	3 C	■	■	4 F	5 O	R
■	I	■	6 T	R	O	F	F	■
7 U	L	D	R	■	■	■	8 F	9 S
■	E	■	10 L	11 I	S	T	■	P
12 U	S	13 A	■	N	■	■	14 O	R
S	■	B	A	S	I	15 C	■	I
16 I	N	S	■	T	■	17 O	C	T
N	■	■	18 T	R	19 O	N	■	E
20 G	O	T	O	■	21 N	T	T	■

X軸方向のカギ

- ●1＝ある国会議員がアメリカで電話をかけて「もしもし」のかわりにこれを2回いったという、笑い話がある
- ●4＝TO、NEXTとチームを組んでループを作る
- ●6＝X軸方向の18番のコマンドで入ったモードを解除するコマンド
- ●7＝DRAW文のデータとして読むと「上左下右」。四角をかくGML
- ●8＝パナソニックA1シリーズの機種名にはかならずこの2文字が付く
- ●10＝リストを表示するコマンド
- ●12＝BASICの生まれた国
- ●14＝どちらかが真なら結果が真になる論理演算子
- ●16＝文字と文字のあいだに新しく文字を追加するときに使うキー
- ●17＝もともと8という意味。このあとに$とカッコをつけると数値を8進数表記の文字列に変換する関数になる
- ●18＝このコマンドを実行しておいて、テキストモードのプログラムを実行すると行番号がずらずら出てくる。
- ●20＝もともと「〜へ行く」という意味。後藤さんではない
- ●21＝パソコン通信をするときにおせわになる巨大企業。むかし公社だった

Y軸方向のカギ

- ●2＝ディスクに入っているプログラムなどの一覧表を表示させるコマンド
- ●3＝このキーを押しながらGキーを押すとビープ音が鳴る
- ●4＝&H___のキャラクタコードを持つキャラクタはカーソル
- ●5＝このことば単独ではBASICに現れないが、SPRITEやINTERVALと組み合わせて出てくる
- ●9＝もともと妖精という意味。細かい動きが得意
- ●11＝ある文字列の何番目に指定した文字列が登場するかを判定する関数
- ●12＝PRINTと組み合わせて、文字の表示の仕方を決める
- ●13＝たとえば5から7を引いたときでも答えを2にしてしまう関数
- ●15＝やめたとちゅうから再開するコマンド。市販のゲームソフトには通用しない
- ●18＝もともと、「〜へ」という意味。FOR〜NEXTループやCOPY文に出てきて、行き先を表す
- ●19＝Y軸方向の5番のことばとは反対の意味を持ち、INTERVALやSPRITEなどのほか、GOTOやKEYなどとも組み合わせられる

問2

【ワードメイズパズル】画面中央下部付近で、ある文(※注参照)を実行したところ、写真のように画面上部に文字がずらりと表示された。下図のある文字から順に縦横斜めの8方向いずれかに文字をつないでいけばその文ができあがる。図にその順路を示せ。

```
I , I + 6 1 4 4 ,
E K : 5 5 2 O T E
F O F 2 O T Ø Ø =
O P E 5 5 I = Ø I
K V : 5 R $ I = R
E I , C H P R I O
, + I T O F O R F
4 6 : X R : = I T
4 1 N E = 5 5 2 O
```

↑中央付近である文(写真では白く隠されている)を実行すると上部にずらりと文字が

※注　その「ある文」は、3つのステートメントを2つのコロン(:)でつないだもの。文字数は総計で25文字。また、カンマ(,)を1個含み、スペースを含まない。

MSX非公式模擬試験問題B

問3

【打ちこみミス発見パズル】右の画面は、左のリスト1(MSX2/2+用)を一気に打ちこんだばかりのところ(LISTによるリスト表示はまだしていない)だが、打ちこみミスが6か所ある。リスト1と写真とでくいちがう文字を、リスト1上で丸で囲んですべて示せ。

■リスト1

```
10 COLOR,0,0:SCREEN6,3
20 SPRITE$(0)=STRING$(32,255)
30 COLOR=(1,7,0,0):COLOR=(2,0,7,0):COLOR
=(3,0,0,7)
40 LINE(50,82)-(113,113),1,BF
50 PUTSPRITE0,(25,49),1,0
60 PUTSPRITE1,(25,113),7,0
70 LINE(114,82)-(177,113),2,BF
80 PUTSPRITE2,(57,49),2,0
90 PUTSPRITE3,(57,113),6,0
100 LINE(178,82)-(241,113),3,BF
110 PUTSPRITE4,(89,49),3,0
120 PUTSPRITE5,(89,113),14,0
130 GOTO130
```

```
10 COLOR,0,0:SCREEN6,3
20 SPRITE$(0)=STRING$(32,255)
30 COLOR=(1,7,0,0):COLOR=(2,0,7,0);COLOR
=(3,0,0,7)
40 LINE(50,82)-(113,113),1,BF
50 PUTSPRITE0,(25,49),1,0
60 PUTSPRITE1,(25,113),7,0
70 LINE(114,82)-(177,113),2,BF
80 PUTSPRITE2,(57,49),2,0
90 PUTSPR1TE3,(57,113).6,0
100 LINE(178,82)=(241,113),3,BF
110 PUTSPRITE4,(89,49),3,0
120 PUTSPRITE5,(89,113),14,0
130 GOTO13O
```

問4

【RUNの結果】下のリスト2(MSX2/2+用)を実行したあとの画面状態を(a)〜(c)のなかから選べ。

■リスト2

```
10 COLOR 15,4,7:SCREEN 0:WIDTH 80:PRINT"
CIRCLE IS COMING! CIRCLE IS COMING!":FOR
 W=0 TO 500:NEXT
20 SCREEN2:OPEN"GRP:"AS#1
30 WIDTH 40:DRAW"BM100,100":PRINT#1,"CIR
CLE":CIRCLE (120,100),50
40 GOTO 40
```

選択肢

(a)

CIRCLE

一瞬、縦長の文字で「CIRCLE IS COMING! CIRCLE IS COMING!」と表示して、円とそのほぼ中央に「CIRCLE」という文字を通常の文字で表示する

(b)

CIRCLE

一瞬、縦長の文字でメッセージを表示し、円を表示するまでは、(a)とまったくおなじだが、円のなかに表示される文字は、最初に表示されたメッセージとおなじような縦長の文字である

(c)

Illegal function call in 30
Ok

一瞬、縦長の文字で「CIRCLE IS COMING! CIRCLE IS COMING!」と表示してから、写真のようなエラーメッセージを出して停止する

問5

【ダイレクトステートメント】(a)〜(g)の文(白い四角はカーソル)を実行した結果、表示される文字をそれぞれ答えよ。ただし「Ok」は含まない。なお、それぞれリセットした直後に実行したものとする。

(a)

```
PRINT HINT■
```

(b)

```
PRINT &B1010■
```

(c)

```
SCREEN 1+1=2■
```

(d)

```
SCREEN E=NOTE■
```

(e)

```
DEFUSR=342■
```

(f)

```
DEFFNA=RND(1)*6+1■
```

(g)

```
Mマカ゛モオモシロイケト゛=100:Mファンハモットオモシ
ロイ!=80:PRINT テ゛ハ、Mファンハナンテンカナ■
```

電源を入れて解答と解説

問1 下図(問1)のとおり。

問2 下図(問2)のとおり。ちなみに、全文をしるすと「FOR I=ØTO255:VPOKEI,I:NEXT」となる。人によっては、VPOKEの直後のIに「+6144(16進数にすると&H18ØØ)」が入るはずだと思った人もいるかもしれないが、それはSCREEN1での話。この写真をよく見ると横の字数は40字だし、文字も6ドットぶんしか表示されていないので、SCREEN0だとわかる。この文は、SCREEN0のパターン名称テーブル(VRAMアドレスは0からはじまる)に、0～255を書きこむことで、グラフィックキャラクタを含む全文字を表示しているのだ。

問3 ちがっているところは、①行30の右端から6文字目の「:」(「;」に打ちミス)②行60のPUTSPRITE1の「1」(小文字の「l(エル)」に打ちミス)③行90のPUTSPRITEの「l」(数字の「1」に打ちミス)④おなじく行90の右端から4番目の「,」(「.」に打ちミス)⑤行100の真ん中付近にある「ー」(「=」に打ちミス)⑥行130の最後の「Ø」(アルファベットの「O」に打ちミス)

ところで、打ちこみミスを修正して、このリスト1を実行すると、右下のような画面になる。このプログラムは、パレットコード(いわゆるカラーコード)が0～3しかないはずのSCREEN6で10色を表示するものなのだが、くわしくは右の傍注参照。ただし、SCREEN6は、SCREEN7が使える機種(ほとんどのMSX2が使える)では実用的価値が薄い。

問4 ⓒ。つまり、エラーになる。ⓒを選んだ人にさらに問いたい。エラーは行30のどの部分で出たのか。次の①～③のなかから選べ。①WIDTH文②DRAW文③PRINT#文。答えは、②。DRAW文以降を行31、32にわけたうえで実行してみるとわかる。

しかし、じつは、エラーの原因を作っているのはWIDTH文で、じっさい行30のWIDTH文を削除するとエラーは出ずにⓐの画面になる。妙な話だ。

じつはこのエラーのもうひとつの原因は行10にある。これとおなじ現象を、3月号のファンダムに掲載した『BATTLE SHOOTING』の「恐怖のWIDTHを発見」というカコミで紹介しておいたが、記憶に残っていただろうか。

問5 ⓐ「Ø」、次の行に「Syntax error」⇒変数Hの内容(0)を表示したあと、INTという予約語が直接続くためエラーになる。ⓑ「5」、2文字あいて「Ø」⇒2進数&B1Ø1を10進数で表示したあと変数Oの内容(0)を表示。ⓒ「Illegal function call」。⇒関係演算式「1+1=2」の値は−1になるため、SCREEN−1を実行しようとしてエラーになる。ⓓなにも表示しない(SCREEN0を実行して画面をクリアする)。⇒関係演算式「E=NOTE」の右辺は否定の論理演算子が変数Eについている形。EがEの否定に等しいことはないので、この関係演算式の値は0。したがってSCREEN0を実行する。ⓔなにも表示しない。⇒この文はキーバッファクリア(先行入力をカットする)をUSR関数に定義するもの。ⓕ「Illegal direct」⇒DEFFN～は、プログラム中でのみ使用できるステートメントで、ダイレクトモードで使用しようとするとこのエラーが出る。ⓖ「1ØØ」⇒プログラム中でもそうだが、REM文やDATA文、文字列以外でのひらがな、カタカナなどは完全に無視される。だから、この文は、「M=1ØØ:M!=8Ø:PRINTM」とおなじことになる。だから、「1ØØ」が表示されるのだ。数値変数は、初期状態では「%」、「!」、「#」(それぞれ、整数型、単精度型、倍精度型)の型宣言文字をつけないと倍精度型とみなされるため、変数Mは倍精度型変数、M!は単精度型変数として別々にあつかわれるのだ。

ちなみによく見られる「DEFINTA-Z」はA～Zではじまる数値変数全体について整数型宣言をするという意味だ。

■問1

I	F	■	C	■	■	F	O	R
■	I	■	T	R	O	F	F	■
U	L	D	R	■	■	■	F	S
■	E	■	L	I	S	T	■	P
U	S	A	■	N	■	■	O	R
S	■	B	A	S	I	C	■	I
I	N	S	■	T	■	O	C	T
N	■	■	T	R	O	N	■	E
G	O	T	O	■	N	T	T	■

■問2

```
I,I+6144,
EK:5520TE
FOF2OTØØ=
OPE55I=ØI
KV:5R$I=R
EI,CHPRIO
,+ITOFORF
46:XR:=IT
41NE=5520
```

SCREEN6　SCREEN6は、パレットコード(いわゆるカラーコード)が0～3しか使えないことになっている(マニュアルをよく読むとどこかに書いてあるはずだ)。じっさい、前景色、背景色の指定に4以上の数値を使っても、自動的に4で割ったあまりをかってに計算して、その数値で色を表示している。しかも、初期状態では、パレットコード0は透明、1は黒、2、3は明るさのちがう緑で、ほとんどなんの色気もない。SCREEN2以上のプログラムをSCREEN6に変えて走らせてみると色気のなさがよくわかる(SCREEN7のプログラム以外は左右のサイズも変わる)。そこで、COLOR=～でパレットを変更するほかないのだが、じつは 周辺色(画面の枠の外にある色)やスプライトだけは、4つのパレットコードのうちから2つをシマ模様に組み合わせて10色ぶんを表示することができるのだ。

リスト1は、パレットコード1～3を赤、緑、青に変更して中段にLINE文のボックスフィルでパレットコード1～3の四角をかき、上段と下段にそれぞれの色の組み合わせ(0～3から異なる2つを取る組み合わせで6とおり)でスプライトを表示している。これで9色。残りはバックの透明(0)だ。じつは透明色もある方法でパレット変更できるが、ここでは手抜きした。

ところで、スプライト(周辺色もおなじ)のパレットコードはふつうのコードではない。ここの数字は2進数2桁を2つ組み合わせたものなのだ。

たとえば、スプライト面番号0の色は1だが、2進数4桁にすると、&BØØØ1。2桁ずつ取ると、0と1。そこで、このスプライトの色は透明(ここでは黒とおなじ)と赤のシマ模様になり、暗い赤に見える。2進数4桁にしたときの上位2桁がグラフィック座標の偶数番目の色を指定し、下位2桁が奇数番目の色を指定しているのだ。

恐怖のWIDTHを発見　3月号のファンダムのくりかえしになるが、SCREEN0、WIDTH41～80のテキストモードで始まるプログラムでは、WIDTH40以下を実行すると、そのときのスクリーンモードがなんであってもSCREEN0にもどってしまうのだ。だから、リスト2の行30のWIDTH40でSCREEN0になり、DRAW文でエラーになるわけだ。

4色しか表示できないSCREEN 6 で10色表示

MSXの神秘数・キャラクタコード27

BASICピクニック

#12うわさのエスケープシーケンス

授業からのエスケープ。現実からのエスケープ。常識からのエスケープ。さまざまなものから逃げていった先に、なにか新しいものが待っている、ような。

エスケープシーケンス escape sequence。この場合のエスケープとは、キャラクタコード27のコントロールキャラクタ、すなわちCHR$(27)のことで、しばしばESCと略される。また、エスケープキャラクタ、拡張文字と呼ばれることもある。
たとえば、リターンキーを押したときはCHR$(13)がキーボードからMSX内部に送られるように、ESCキーを押したときにはこのCHR$(27)が送られるのだが、MSXではESCキーを押してもなにも起こらない。ところが、PRINT文中でCHR$(27)が出てくると、それに続く特定の文字が特別な意味をおびるようになる。これこそが、エスケープキャラクタのオリジナルな使われ方で、エスケープとそれに続く本来の意味を変えられた文字のつながりをエスケープシーケンスという。
もともと、エスケープということばは「逃避、脱走」という意味だが、エスケープキャラクタによって本来の文字の意味から脱走して、特別な意味を持たせるというようなところからこう名づけられたのだろう。
ダブルクォーテーションマーク(″) この文字は文字列を囲む特殊な文字なので、文字列そのものには入れることができない。このためにX座標かY座標が2になっている座標をエスケープシーケンスで指定するときに困ってしまう。そこで、たいていはその座標を避けているが、やむをえない場合は、CHR$(34)、つまりダブルクォーテーションマークのキャラクタコードを使って指定することになる。

LOCATE文のないプログラム

写真は、リスト1のプログラムを実行している画面だ。ビリケンヘッドの男の子の口がカーソルキーの入力に応じて3つのパターンに変わる。

顔の形に並べられている文字は「*」。行40のPRINT文に続く文字列の最後にダブルクォーテーションマークにはさまれてちょこっと顔を出している。画面に表示されている「*」は、すべてこの「*」だ。

このプログラムでは、まったくLOCATE文を使っていない。LOCATE文のかわりに座標指定の機能を発揮しているのが「*」のまえにある「E$～」なのだ。これが、ファンダムでもよく登場する座標指定のエスケープシーケンスだ。

エスケープシーケンスはこれ以外にもいくつか種類がある。その不思議な機能と使い方が、今月のピクニックのテーマだ。

```
10 COLOR 1,12:SCREEN 0:WIDTH 40:KEYOFF
20 FOR I=0 TO 5:READ A$(I):L(I)=LEN(A$(I
))¥2:NEXT:A$(7)=A$(3):L(7)=L(3):E$=CHR$(
27)+"Y":A$=A$(0)+A$(1):L=L(0)+L(1)
30 FOR I=1 TO L
40 PRINT E$;MID$(A$,I*2-1,2);"*":NEXT
50 S=STICK(0):IF S MOD 2 =0 THEN 50
60 FOR I=0 TO 4
70 PRINT E$;CHR$(44+I);"/";SPC(9):NEXT
80 A$=A$(S):L=L(S):GOTO 30
90 DATA "5.4/4031201/1.0-0,/+.*-*,)+)*))
*(*'+&,&-%.%/%/%0$1$2#3$4$5%6%7%8&9&:';(
<)<*=+=,=-<.</;0:0918172635464758*/*7","
,3-4.5/403/2.1-2",,"-/.0.1/2/3/4.5.6-7",
,"./-0.1/2.3-4.5/6.7"
```

↩プログラムのくわしい解説は右のページで

リスト1：顔のアニメーション

↑初期画面。カーソルキーの上を押してもこの顔になる

↑カーソルキーの右か左を押すと「ニコッ」

↑カーソルキーの下を押すと口元がげげっとなる

座標指定機能を持つ文字

通常のLOCATE文……LOCATE X, Y

エスケープシーケンス……CHS$(27);"Y";CHR$(32+Y);CHR$(32+X);

※X座標とY座標が逆の順序になる

エスケープシーケンスに使用する文字と座標の関係

! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G

0 10 20 30

! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7

0 10 20

※「"」は文字列中では使えないので、どうしても使いたい場合はCHR$(34)の形で使うこと。
また、座標が0になるときはX、Y座標とも空白が対応。

じっさいに現れる形
CHR$(27);" Y5.";

CHR$(32+14) CHR$(32+21)

座標(14, 21)

PRINT CHR$(27);"Y5. *"
と打ちこんでリターンキーを押してみよう。画面の上のほうでやったほうがわかりやすいだろう。これを実行すると、「*」が画面の下のほう、座標(14, 21)の場所に表示される。Yのあとの文字を変えると表示場所も変わる。

PRINT文の文字列のなかに、CHR$(27)と、その直後に「Y」(かならず大文字)があると、そのあと2文字(ここでは「5.」)はふつうの文字ではなく、座標指定の文字とみなされて、キャラクタコードに応じた位置(キャラクタコード=32+座標)にカーソルが移動するのだ。そして、それに続く文字(ここでは「*」)がふつうの文字としてその場所に表示される、という仕組みだ。

座標指定用として使われるときの文字と座標の関係をまとめて左の図に示した。じっさいに使うときは、Y座標、X座標の順にならべるという点に注意(LOCATE文とは逆)。

こんがらかったリスト1を解きほぐすと

10　画面の初期設定
20　座標データ群をA$(n)に読みこみ、それぞれの座標データの個数をL(n)に設定/A$(7)、L(7)の設定/E$にCHR$(27)+"Y"を入れる/初期画面を表示するための変数の設定
　A$(n)……A$(0)は、顔の外形と目の座標データ群。A$(1)、A$(3)、A$(5)は口の部分で、それぞれスティック入力が1(上)、3(右)、5(下)のときの座標データ群。A$(7)、つまりカーソルキーの左を押したときのデータは、右を押したときのA$(3)をそのまま使用。また、A$(2)やA$(4)なども形式上読みこまれているが、なにも設定されない
　L(n)……座標データ群が持つ座標の個数。1つの座標は2つの文字でできているので、L(n)はA$(n)の長さの半分になる。行40のループの終値Lにそのつど設定される
　E$……座標指定のエスケープシーケンスの頭の部分。ここでは座標指定だけなので、CHR$(27)と"Y"を組にして設定
30〜40　顔または口の表示⇨E$以降に続くMID$で取り出される2文字が座標データ。データに応じた座標に「*」を表示していく
　L……ループの終値(座標の個数)
　A$……そのつど使用する座標データ群。MID$で2文字ずつ取り出して使用。A$、Lとも、行80でスティック入力の値に応じて設定される(ただし最初は行20で設定)
50　スティック入力⇨4方向のみ受け付け、それ以外(入力値が偶数のとき)はくりかえし
60〜70　口の部分の消去⇨口を表示する部分にスペースを9個ずつ表示してきれいにしておく。ここは、むしろLOCATE文を使ったほうが短いが、サンプルとしての意地を貫いてあえてエスケープシーケンスでやった
80　行50で受け付けたスティック入力の値に応じたデータをA$、Lに設定して行30に飛ぶ
90　座標データ⇨A$(0)〜A$(5)の順に文字データをダブルクォーテーションマーク(")でくくってならべている。LOCATE文を使うとここのデータが長くなる(右のリスト参照。データ以外はだいたいリスト1とおなじ構造)。とちゅうの連続するコンマは、不要なA$(2)、A$(4)用の空データのため

参考:リスト1と同じことをするプログラム

```
10 COLOR 1,12:SCREEN 0:WIDTH 40:KEYOFF
20 READ X,Y
30 IF X=0 THEN 50
40 LOCATE X,Y:PRINT "*":GOTO 20
50 S=STICK(0):IF S=1 THEN RESTORE 110 EL
SE IF S=3 OR S=7 THEN RESTORE 120 ELSE I
F S=5 THEN RESTORE 130 ELSE 50
60 FOR I=0 TO 4
70 LOCATE 15,12+I:PRINT SPC(9):NEXT
80 GOTO 20
90 DATA 14,21,15,20,16,20,17,19,16,18,15
,17,14,17,13,16,12,16,11,15,10,14,10,13,
9,12,9,11,9,10,10,9,10,8,11,7,12,6,13,6,
14,5,15,5,15,5,16,5,17,4,18,4,19,3
100 DATA 20,4,21,4,22,5,23,5,24,5,25,6,2
6,6,27,7,28,8,28,9,29,10,29,11,29,12,28,
13,28,14,27,15,26,16,25,16,24,17,23,17,2
2,18,21,19,22,20,23,20,24,21,15,10,23,10
110 DATA 19,12,20,13,21,14,20,15,19,16,1
8,15,17,14,18,13,,
120 DATA 15,13,16,14,17,14,18,15,19,15,2
0,15,21,14,22,14,23,13,,
130 DATA  15,14,16,13,17,14,18,15,19,14,
20,13,21,14,22,15,23,14,,
```

セミコロン(;)を省略できる場合とできない場合　今回のプログラムでは、文字と文字変数などのあいだに「;」をかならずつけているが、これは、文字の区切りが見やすいようにつけたもので、じつは、ほとんどが省略できる。つまり、

PRINT E$;″E″;

は、

PRINT E$″E″;

とまったくおなじなのだ。ただし、文字列の最後の「;」だけは省略できない。これは、PRINT文に続く文字列の最後に「;」がない場合は、自動的にカーソルのある位置から次の行の先頭にカーソルが移動してしまうため、あるのとないのとでは動きがかなりちがってくるためだ。たとえば、

PRINT CHR$(27)″E″;

と

PRINT CHR$(27)″E″

を2つとも実行してどうなるかを見てみるとよくわかる。前者は、画面をクリアして画面のいちばん上の行に「Ok」を表示するが、後者は、画面をクリアして2番目の行に「Ok」を表示する。このセミコロンの落とし穴にはよく落ちるので、あってはならない場合以外、文字列の最後にはセミコロンをつけておいたほうが安全。

セミコロンと「+」　セミコロンで連結している文字のつながり全体を1つの文字変数に入れたい場合は、セミコロンのままではエラーになる。この場合は、セミコロンのかわりに「+」を使う。たとえば、

CHR$(27);″L″

を1つの文字変数にしたいときは、

A$=CHR$(27)+″L″

として、

PRINT A$;

のようにして使う(この場合は、その行以降が1行下にスクロールする)。

いろいろなエスケープシーケンス

エスケープシーケンスには、下の表のようにぜんぶで17種類ある。大きくわけると、①カーソルを移動させるもの6種類、②画面クリアを含め文字を消去するもの5種類、③画面をスクロールさせるもの2種類、④カーソル表示を変えるもの4種類、の4つにわけられる。座標指定のエスケープシーケンスは、カーソルを移動させる6種類のうちの1つにすぎないのだ。

座標指定のエスケープシーケンスだけは座標を指定する文字のぶんだけちょっと複雑だが、エスケープシーケンスの使い方は基本的にみなおなじで、CHR$(27)の直後に、それぞれの機能を指定する文字を下の表のようにつなぐだけだ。だいじなのは、大文字と小文字をきちんと区別すること。

また、エスケープシーケンスの終わりは自動的に判断されるので、書式さえ正しければこのエスケープシーケンスの文字列のあとに表示したい文字をてきとうにつないでかまわない。

ではそれぞれの解説。

①カーソルを移動させるもの

カーソル移動のエスケープシーケンスは6種類あるが、座標指定のもの以外は、多くの場合、備考の欄に書いてあるようなコントロールキャラクタで置き換えることができるため、あまり使われることがない。

②文字を消去するもの

画面をクリアするエスケープシーケンスは2種類あるが、これは2つともまったくおなじ機能で、しかも、これもコントロールキャラクタ1文字におなじ機能を持つものがあるので、あまり使われない。

しかし、「E$;"K"」と、「E$;″J″」は、エスケープシーケンスにしかない機能で、ゲームプログラムでの応用が期待できるだろう。じっさいに、「J」のほうは、スクロール式シューティングゲームで「あるアイテムを取るとそれ以降のキャラクタが一瞬にして消える」という形でいかされた例もある(1988年8月号掲載の河内和彦作『DISTANCE』)。

また、「E$;″l″」(小文字のエル)は、CTRL+Uキーを押したときの働きに似ている。カーソルがどこにあっても行全体を消去するのだ。ただし、「M」のようにスクロールはしない。あくまで、カーソルのある行の左から右端までを消すのだ。

③画面をスクロールさせるもの

「L」と「M」は、機能欄で1行挿入、1行削除となっているが、それよりも下スクロール、上スクロールという機能のほうが目玉になる。これもエスケープシーケンス独特の機能で長いプログラムなどでもよく使われる。31ページのリスト2参照。

④カーソル表示を変えるもの

カーソル表示、カーソル消去の2つはLOCATE文の第3パラメータ(カーソルスイッチ)と機能はおなじだが、残りはエスケープシーケンス独特のものだ。ちょっと特殊なのでどのように使っていいか迷うが、ためしてみるとけっこう楽しい。31ページのリスト3参照。

エスケープシーケンス一覧表

※ただし、表中のE$はCHR$(27)を表す

エスケープシーケンスの種類	機　能	備　考
E$;"A"	カーソルを上に移動	CHR$(30)と同様の機能
E$;"B"	カーソルを下に移動	CHR$(31)と同様の機能
E$;"C"	カーソルを右に移動	CHR$(28)と同様の機能
E$;"D"	カーソルを左に移動	CHR$(29)と同様の機能
E$;"H"	カーソルをホームポジションに移動	CHR$(11)と同様の機能
E$;"Y";CHR$(32+Y座標);CHR$(32+X座標)	カーソルを(X座標、Y座標)の位置に移動	LOCATE文と同様の機能
E$;"j"またはE$;"E"	画面をクリア	CHR$(12)と同様の機能
E$;"l" 注・小文字のエル	1行すべてを削除	CTRL+Uキーに近い
E$;"K"	行の終わりまでを削除	カーソル以降を削除
E$;"J"	画面の終わりまでを削除	カーソル以降を削除
E$;"L"	1行挿入	その行以降は下へスクロール
E$;"M"	1行削除	その行の次行以降は上へスクロール
E$;"x4"	カーソルの形を「■」にする	
E$;"x5"	カーソルを消す	LOCATE,,0と同様の機能
E$;"y4"	カーソルの形を「▂」にする	いわゆる挿入モードにはならない
E$;"y5"	カーソルを表示する	LOCATE,,1と同様の機能

⇧エスケープシーケンスを試すときはかならずPRINT文のあとにエスケープシーケンスを置き、文字列全体の最後に「;」をつけて実行すること

スクロール機能を持つエスケープシーケンス

リスト2：赤いピラミッドが上下に動く

```
10 COLOR 8,4:SCREEN 0:WIDTH 40:KEYOFF
20 E$=CHR$(27):LOCATE,,1:LOCATE,,0
30 FOR I=0 TO 7
40 LOCATE 19-I,12+I:PRINT STRING$(I*2+1,
255)
50 NEXT
60 LOCATE 0,0
70 FOR I=0 TO 10:PRINT E$;"M";:NEXT
80 FOR I=0 TO 10:PRINT E$;"L";:NEXT
90 GOTO 70
```

←赤いピラミッドがスムーズに上下に動く

リスト２は、赤いピラミッドを11文字の幅で上下に揺らすプログラムだ。プログラム中、LOCATE文のところは、すべてエスケープシーケンスに置き換えることができるが、ここではリスト１とちがって意地をはらずにわかりやすいほうを使った。

行20は、E$にCHR$(27)を設定するついでに、LOCATE文を使ってカーソル表示とカーソル消去をおこなっている。少なくとも、カーソル表示をやっておかないと、カーソルキャラクタで構成されているピラミッドが見えなくなるのだ。

行30〜50は、ピラミッドの形にカーソルキャラクタを表示する部分。行60〜行90は最上行で１行挿入、１行削除をおこなうことで、ピラミッドの形をそのまま上下に揺らしているわけだ。

この動きとおなじことをエスケープシーケンスを使わずにやるのはかなりめんどうだ。部分的にかきたし、よけいな部分を空白で消せばいいのだが、プログラムが複雑になってしまううえに、実行スピードもかなりおそくなってしまう。ひまな人は、右の参考リストを打ちこんで実行してみて、リスト２の動きと比べてみてほしい。

参考：かきかえでやるとすごくおそい

```
10 COLOR 8,4:SCREEN 0:WIDTH 40:KEYOFF
20 LOCATE,,1:LOCATE,,0
30 FOR I=0 TO 7
40 LOCATE 19-I,12+I:PRINT STRING$(I*2+1,
255)
50 NEXT
60 FOR I=0 TO 10:V=459-40*I:FOR J=0 TO 7
:VPOKE V+39*J,255:VPOKE V+41*J,255:NEXT:
LOCATE 12,19-I:PRINT SPC(15):NEXT:FOR I=
0 TO 10:LOCATE 12,9+I:PRINT STRING$(15,2
55):V=59+40*I:FOR J=7 TO 0 STEP-1:VPOKE
V+39*J,32:VPOKE V+41*J,32:NEXT:NEXT
70 GOTO 60
```

カーソルを変えるエスケープシーケンス

リスト３：どれがホンモノのカーソル？

```
10 COLOR 15,1:SCREEN 0
20 E$=CHR$(27)
30 PRINT E$;"Y)*";"0123456789";
40 PRINT E$;"Y**";STRING$(10,255);
50 PRINT E$;"Y*";CHR$(42+RND(1)*10);"A";
E$;"D";
60 PRINT E$;"y5";:GOSUB 110
70 PRINT E$;"y4";:GOSUB 110
80 PRINT E$;"x4";:GOSUB 110
90 PRINT E$;"x5";:GOSUB 110
100 GOTO 40
110 FOR W=0 TO 500:NEXT:RETURN
```


↑「Ａ」の反転パターンがデモったあと、ほんとうのカーソルの位置だけがふつうの「Ａ」になる

カーソルは重なった文字を反転させたパターンになるという奇妙な性質がある。ただし、プログラム実行中は、カーソル表示のモードになっていないとそうはならない。

このプログラムでは、行30、40で、０〜９の数字とそのすぐ下にカーソルキャラクタ10個を表示しておき、行50でそのカーソルキャラクタ群のどこかに「Ａ」を表示し、カーソル位置を１個もどしている（すべてエスケープシーケンス）。「Ａ」を表示してカーソル位置をもどすということはつまりカーソルの現在位置を「Ａ」に重ねるということだ。そこでエスケープシーケンスのカーソル表示をおこなう（行60）と、カーソルキャラクタすべてが「Ａ」の反転パターンになる。そして、カーソルの形を３分の１にし（行70）、またもとの四角にもどすと（行80）、10個の「Ａ」の反転パターンが身をすくめて立ち上がるように変化していく。そして最後にカーソル消去（行90）。このときにようやく、ほんとうのカーソルの位置がはっきりする。つまり、ほんとうのカーソルの位置だけがふつうの「Ａ」の文字になるのだ。リスト３は、これをなんどもくりかえすプログラムなので（「Ａ」の位置は毎回ランダムに変わる）、じっと見ていて、次にどこがほんとうのカーソルの位置なのかをいいあてるゲームにもなる。じつは、ほんとうのカーソル位置は動作中にチカッとひらめくのだ。ただし、よく見ていないと見逃すこともあるくらいのひらめきなので、リスト３は、集中力を養うトレーニングゲームということもできると思う、んだけど。

表示のかわりに制御する「文字」たち

BASICテクニック

#13 コントロールコード

コントロールコードといってもリモコン式戦闘ロボットとコントローラをつなぐコードのことではない。そんなものがあったら空を飛ぶとき困るじゃないか。

CTRLキーの使い道

コントロールコードは、じつはなにかしらやっているときにしょっちゅう顔を出しているものだ。たとえば、リターンキーを押すとコントロールコード13、CLSキー(SHIFT+CLS／HOME)を押すとコントロールコード12がキーボードからMSXのなかに送りこまれる。上下左右のカーソルキーも、30、31、29、28のコントロールコードを送りこむキーだ。

これは、Aのキーを押すとAのキャラクタコード65がMSXのなかに送りこまれるのとまったくおなじだ。

MSXのなかに入ると、コントロールコード以外はディスプレイにその文字が表示されるが、コントロールコードは、文字の表示のかわりに、改行したり、画面を消したり、カーソルを動かしたり、コードに応じたそれぞれの働きをする。

■CTRLキーを活用しよう

コントロールコードを送りこむキーは、先にあげたキーのほか、HOME(CLS／HOMEを単独で押したとき)、TAB、INS、DEL、ESC、SELECTの計13種類。このうちSELECT(コード24)は特別な機能を持っていないので右ページの表では省略した。

0～31のコントロールコードのなかには、ほかにも、とりあえずなんの機能も持っていないものがある(これも右ページの表では省略した)が、特殊キーで入力できるもの以外にも、便利な機能を持つものがいくつかある。それを入力するためのキーがCTRLキーなのだ。とくに、CTRLとB、E、F、G、N、Uなどとの組み合わせで入力されるコントロールコードは便利で重要だ。

また、たとえば、SHIFT+CLS／HOMEのかわりに、CTRL+Lを使ったり、BSのかわりにCTRL+Hを使ったりすることもできる。

じっさい、CTRLキーを使ったやり方なら、いわゆる「ホームポジション」(効率的にキー入力するための指の配置)をくずさずにキーを打っていくことができるので、特殊キーのうちいくつかはCTRLキー方式に置き換えてみるといいだろう。

ところで、右ページの表で、重要なコントロールコードが1つだけ抜けている。コントロールコード127(DELキーに対応)だ。コードが飛んでいて、いかにも特殊な感じがあるが、CTRLキーを使った入力はできないし、PRINT文中で使ったときもちょっとちがう機能になるなど妙なことが多いため、右の表とはべつに、104ページの下に掲載した。

ホームポジション　102ページの本文中に出てくる「ホームポジション」と103ページの表中に出てくる「ホームポジション」とは意味がちがう。表のほうは、画面上テキスト座標(0, 0)の位置のことだが、本文中のホームポジションは、「A」、「S」、「D」、「F」、「J」、「K」、「L」、「;」のそれぞれのキーに左手の小指から人さし指、右手の人さし指から小指をこの順に配置することを意味している。この配置は8本の指をフルに使ってキーをたたいていくときの基本で、つねにこのホームポジションに指をもどすように心がけていると、キーを見ないでも打てるようになる。キーボードによっては「F」と「J」のキーに小さな突起がついていたり、ほかのキーよりも大きなカーブがついていたりすることがあるが、これはこの2つのキーがホームポジションの両手の人さし指の位置に対応しているからだ。そういうキーボードでは手探りでホームポジションであることを確認できる。ただし、最近のMSX2+でこうなっているのはソニーの機種だけのようだ。

コントロールコード表の注

※1　たとえば、

```
PRINT CHR$(1)+"A"
```

を実行すると、グラフィックキャラクタの「月」が表示される。つまり、CHR$(1)は直後のキャラクタ1つをグラフィックキャラクタに変えてしまう働きを持っているわけだ。機能欄でグラフィックヘッダといっているのはこのことだ。ただし、グラフィックヘッダに続く文字はキャラクタコード64～95の範囲に限る。

使えるコントロールコード表

対応キー	キャラクタコード	機能（または対応する特殊キー）	PRINT文中の機能
CTRL＋A	1（&H1）	グラフィックヘッダ（グラフィックキャラクタを取り扱うときに先頭につけるコード）	グラフィックキャラクタの表示に使用 ※1
CTRL＋B	2（&H2）	カーソルを直前の語（※2）の先頭へ移動	
CTRL＋C	3（&H3）	ＩＮＰＵＴ文などの入力待ち状態を終了する（ＣＴＲＬ＋ＳＴＯＰで代用できるので使われることはまずない）	
CTRL＋E	5（&H5）	カーソル以下、行（※3）の終わりまでを削除	
CTRL＋F	6（&H6）	カーソルを次の語の先頭へ移動	
CTRL＋G	7（&H7）	ビープ音を鳴らす（ＢＥＥＰ文とおなじ）	ビープ音を鳴らす
CTRL＋H	8（&H8）	カーソルの１つまえの文字を削除してカーソル以降を１字つめる（ＢＳキーとおなじ）	
CTRL＋I	9（&H9）	次の水平タブ位置へ移動（ＴＡＢキーとおなじ）	次の水平タブ位置までカーソルを移動
CTRL＋J	10（&HA）	行送り（カーソルキーの下に似ているが、最下行でこのコントロールコードを入力すると画面全体が上にスクロールする）	ＰＲＩＮＴ文のセミコロンなしの場合とおなじ
CTRL＋K	11（&HB）	カーソルをホームポジションにもどす（ＨＯＭＥキーとおなじ）	カーソルを（0，0）に移動
CTRL＋L	12（&HC）	画面をクリアし、カーソルをホームポジションにもどす（ＣＬＳキーとおなじ）	ＣＬＳ文とおなじ
CTRL＋M	13（&HD）	カーソルを左端にもどし（ＲＥＴＵＲＮキーとおなじ）、次の行の先頭へ	カーソルを左端にもどす（行送りはしない）
CTRL＋N	14（&HE）	カーソルを行末へ移動	
CTRL＋R	18（&H12）	挿入モードのオン／オフ切り換え（ＩＮＳキーとおなじ）	
CTRL＋U	21（&H15）	カーソルのある行の先頭から終わりまでをすべて削除（カーソルがどこにあってもおなじ）し、カーソルを行の先頭がもとあった位置まで移動	
CTRL＋[	27（&H1B）	なにもしない（ＥＳＣキーとおなじ）※4	エスケープシーケンスの先頭に使用
CTRL＋¥	28（&H1C）	カーソルを右へ移動（カーソルキーの右とおなじ）	カーソルを右へ移動
CTRL＋]	29（&H1D）	カーソルを左へ移動（カーソルキーの左とおなじ）	カーソルを左へ移動
CTRL＋^	30（&H1E）	カーソルを上へ移動（カーソルキーの上とおなじ）	カーソルを上へ移動
CTRL＋_ ※5	31（&H1F）	カーソルを下へ移動（カーソルキーの下とおなじ）	カーソルを下へ移動

表中※1〜※5は102、104ページの傍注参照。また、ピンクの欄は重要な機能のうちCTRLキーでしかキー入力できないもの。緑の欄はPRINT文中で使用できるもの

コントロールコード表の注(続き)

※2　ここでいう「語」とは、英数字だけでできた文字列のこと。つまり、英数字以外の記号があれば(スペースも含めて)そこを「語」の切れ目とみなす。

※3　ここでいう「行」とはたんに見かけ上の行ではなく、プログラムの行のように、ひとつながりになっている文字のまとまりのこと。画面にいろんな文字がある場合、どこまでが1つの「行」になっているかがわかりにくいことがあるが、BSキーを押し続けても止まってしまうところが行のはじめ、DELキーを押したときに影響のおよぶ最後の見かけ上の行が行の終わりだ。

※4　ESCキーはとくになんの機能もない。しかし、たとえば、
PRINT　INPUT$(2);
を実行して(入力待ちの状態になる)、ESCキー、L、という順に押すとエスケープシーケンスの1行挿入が起こる。この現象は、INPUT$(n)という関数のせい。この関数を含む文を実行すると、n回キーが押されるまで入力待ちを続け、終わった時点でINPUT$(n)がそれまで押されたキーに対応した文字列になるのだ。
ここでは、入力を終えたときに、CHR$(27)+"L"という文字列になっているわけだ。

※5　右のSHIFTキーのすぐ左にあるキーは、アルファベットモードのときに単独で押してもなにも文字を表示しないが、SHIFTキーといっしょに押すと、この「_」(アンダーバー)を表示する。アンダーバーは、「CALL」の略にも使える重要なものなのだが、なぜこんなに打ちこみにくい場所にあるのかわからない。それに、CTRLキーを使ったコントロールコード入力のなかで、唯一、コード31だけはこの記号のせいで3つのキーを押さなくてはならなくなっている。徹底的にCTRLキーを使いたい人は、カーソルキー下のかわりにほぼ機能がおなじCTRL+Jを使うといい。

リスト1の補足

F1キー　ファイル一覧表は、かならず13文字ごとにファイル名の先頭がくるようになっている。だから、カーソルを左端に置いて、リスト1で定義されたF1キーを押すとその行の2番目にあったファイル名がカーソルのところまでやってくる。そこでF2キーを押すと、ファイル名を打ちこむことなく、目的のファイルをロードできる。

CHR$(34)　ダブルクォーテーションマーク「"」は、直接文字列中の文字としては入力できないため、こういう場合CHR$関数を使う。

CTRL+U　画面にいろいろな文字が表示されたままの状態でなにかの命令を入力、実行しようとすると、画面に残っていた文字がなにかの理由で打ちこんだばかりの文字とつながってエラーになることがある。命令を打ちこむまえにCTRL+Uをしておくとそういうことはなくなる。

PRINT文中のコントロールキャラクタ

■ファンクションキーで

コントロールコードは、CHR$関数やSTRING$関数を使って、ファンクションキーの定義に使うことができる。

たとえば、「FILES」と打ちこみ、リターンキーを押してファイル一覧を表示させ、ファイル名の先頭にカーソルをあわせてINSキーを押し、「LOAD"」と打ちこんでリターンキーを押す、といった作業を、下のリスト1によって定義されたファンクションキーなら、F1～F3とカーソルキーの下を使うだけですむ。

ほかにもいろいろな使い方が考えられるので、じっくりコントロールコードを研究して、自分にとっていちばん使いやすいファンクションキーを作ってみるといい。

■PRINT文のなかで

おもしろいことに、コントロールコードのなかには、ふつうの文字のようにPRINT文中に使って機能を発揮させられるものがある。ここでもCHR$やSTRING$を使う。

グラフィックヘッダのコード1、エスケープシーケンスのコード27はちょっと特殊だが、次の8種類のコントロールコードがいろいろと使えて楽しい。

①ビープ音を鳴らすコード7
②タブ移動のコード9
③行送りのコード10
④カーソルホームのコード11
⑤CLSのコード12
⑥行の先頭にもどるコード13
⑦カーソル移動のコード28～31
⑧カーソルをまえにもどしながら文字を消すコード127

このうち、ゲームプログラムでよく使われているのが、⑦のカーソル移動のコントロールコードだ。これを利用すると、いくつかのキャラクタを組みあわせた1つの大きなキャラクタを作ることができる。右ページのリスト2がその例で、変数P$を3×3の大きなキャラクタとしてあつかっている。

⑦以外のコードも同様にしてPRINTされる文字列のなかで機能を発揮する。たとえば、次のような働きをするプログラを考えてみよう。

①画面消去②ビープ音③画面の最下行のX座標8のところで1行ずつ上にスクロールしながらおなじ位置にメッセージ1～3を次々に表示

コントロールコードを使わずにこの動きをプログラミングするには、CLS、BEEPに加えて、LOCATEとPRINTのセットがメッセージの数だけ必要だ。FOR～NEXTとMID$関数などを使えば1セットですませることもできるが、いずれにしてもいくつかの命令文を組み合わせることになる。

それがコントロールコードを使うとたった1つのPRINT文で①～③の働きを実現できるのだ。実例を示しておこう。

PRINT　CHR$(12)CHR$(7)STRING$(23, 1Ø)CHR$(9)"メッセージ1"CHR$(13)CHR$(1Ø)CHR$(9)"メッセージ2"CHR$(13)CHR$(1Ø)CHR$(9)"メッセージ3"

100字以上あるが、これをじっさいに打ちこんでリターンキーを押すと、たしかに①～③のとおりに動く。

つまり、PRINT文以下に続く文字の連なりのなかにCLSやBEEPなどの機能が封じこめられているわけだ。

効果的に使えばプログラムを短くしたり、構造的に単純なものにすることができるだろう。

リスト1：くふうをこらしたファンクションキー作り

■リスト1実行後のF1～F10キー

F1=カーソル以降を13文字ぶんつめる
F2=挿入モードにして「LOAD"」リターン。CHR$(34)は「"」のこと
F3=CTRL+U(以下、Uと略)、「FILES」リターン
F4=U、「LIST」
F5=HOME、CTRL+E、「RUN」リターン
F6=Uとカラー初期設定
F7=U、ビープ5回、「SAVE"」
F8=U、「WIDTH4Ø」リターン
F9=CLS、「LIST.」のあと表示された行の文末にカーソルを移動
F10=CLSとスクリーン初期設定

```
10 KEY1,STRING$(13,127)
20 KEY2,CHR$(18)+"LOAD"+CHR$(34)
30 KEY3,CHR$(21)+"FILES"+CHR$(13)
40 KEY4,CHR$(21)+"LIST"
50 KEY5,CHR$(11)+CHR$(5)+"RUN"+CHR$(13)
60 KEY6,CHR$(21)+"COLOR15,4,7"+CHR$(13)
70 KEY7,CHR$(21)+STRING$(5,7)+"SAVE"+CHR$(34)
80 KEY8,CHR$(21)+"WIDTH40"+CHR$(13)
90 KEY9,CHR$(12)+"LIST."+CHR$(13)+STRING$(2,2)+CHR$(14)
100 KEY10,CHR$(12)+"SCREEN0:KEYON"+CHR$(13)
```

はぐれコントロールコード DELETE(デリート)

対応キー	キャラクタコード	機能(または対応する特殊キー)	PRINT文中の機能
DEL (このキーのみ対応)	127(&H7F)	カーソルの指す文字を削除し、以降の文字をまえにつめる	カーソルを1つまえにもどし、そこにある文字を消去(DELキーの機能とはちょっとちがう)

大きなキャラクタを動かす

【リスト2の動き】リスト2を実行すると、「a」〜「i」が3×3にならんだまま動きはじめる。画面の端では等角度反射ではねかえりながら、ずっと動いていく。

【リスト2の解説】行20でC$に設定しているのが、カーソル移動のコントロールコードだ。CHR$(31)は「カーソルを下に移動」、STRING$(3, 29)はCHR$(29)を3個ぶんという意味で「カーソルを左に3つ移動」。つまり、C$は全体で「カーソルを下に1つ、左に3つ移動」という働きを持っていることになる。

このC$をノリのように使って「abc」「def」「ghi」をくっつけているのが、行30。ここでP$に設定される文字列が、リスト2の中心テーマである3×3の大きなキャラクタだ。

下にある図「3×3キャラクタの仕組み」を見てほしい。文字を表示するたびにカーソルは1つずつ右に動いていくので、「abc」を表示しおわった直後のカーソルは「c」の右にある。そこでC$によってカーソルを「a」の真下に移動させる。この位置で「def」を表示し、ふたたびC$でカーソルを「d」の真下に移動。最後に「ghi」を表示する。

これが、P$を3×3のキャラクタにしている仕組みだ。

行40〜行60はごくふつうの移動計算。行60の「A=SIN(1)」は、計算結果には意味がなく、計算にかかる時間に意味がある。SIN(1)の計算は、だいたい空ループ50回くらいの時間待ち効果があるのだ。

【リスト3について】リスト2のあとにリスト3を追加して打ちこんでみよう(行70は一部修正)。リスト2ではまえに表示したP$を消すために行70でCLSを使っているため、ほかのキャラクタをまわりに表示することができなくなってしまう。そこで、D$に、まえに表示したP$を消去する文字列を作り、行70のCLSのかわりに入れてみた。これで、P$以外にも文字を表示しておくことが可能になる。

【リスト4について】ただの文字では味気ないので「a」〜「i」をパターン定義して、P$が顔に見えるようにしてみた。リスト2、3に追加してみよう。

図 3×3キャラクタの仕組み

リスト2：3×3のキャラクタを動かす

```
10 COLOR 1,7,0:SCREEN 1:WIDTH 32:KEYOFF:
DEFINT A-Z:X=10:Y=10:HX=1:HY=1
20 C$=CHR$(31)+STRING$(3,29)
30 P$="abc"+C$+"def"+C$+"ghi"
40 IF X+HX>29 OR X+HX<0 THEN HX=-HX
50 IF Y+HY>21 OR Y+HY<0 THEN HY=-HY
60 X=X+HX:Y=Y+HY:A=SIN(1)
70 CLS:LOCATE X,Y:PRINT P$;
80 GOTO 40
```

リスト3：「CLS」を不要にする追加リスト

```
35 D$=STRING$(3,127)+CHR$(30)+SPACE$(3)+
CHR$(30)+STRING$(3,127)
70 PRINT D$;:LOCATE X,Y:PRINT P$;
```

リスト4：大きな顔を動かす追加リスト

```
15 FOR I=0 TO 71:VPOKE 776+I,VAL("&H"+MI
D$("0000030E1830203F017EC709020000832048
90E0709808F8409CA2AA9A81413EC6386C448293
294404728AAA620204F820110804020408308201
82442810000000810204080402018",I*2+1,2)):
NEXT
```


↑3×3キャラクタが動く(リスト2)

↑リスト2+リスト4(リスト3を加えてもおなじ)。この顔は担当者の似顔絵

MSX2のBASICで遊ぶ小さなCG

BASICピクニック

＃14 グラフィック遊び

CGツールを使ってサクサクッと絵をかいてもなにかがひと味たりない。では、コツコツと1ドットずつのお絵描きをプログラムでやってみよう。

CG　Computer Graphic(コンピュータグラフィック)の略。C.G.などとピリオドをつけて書くこともあるが、最近ではCGと書くほうが多いようだ。かつては特殊な技術だったのでCGというだけで偉かった時代もあったが、いまではあたりまえすぎてCGということば自体が死語になりつつあるのではないかという感じすらしている。
ドット　点のこと。ディスプレイ上の最小単位のことをドットという。基本的にどんなによくできたCGもドットの集まりであり、ドット絵をきわめればあらゆるCGがかける(はず)。
カラーパレット　この記事では、色を指定するコードのことをカラーコードと呼んでいるが、じつはMSX2以上の機種にはカラーパレット機能があり、あらかじめ決められているコードと色の対応を自由に決められる(512色中16色)ので、絵をかいたあとで色を変えるのも楽しい。

ドットの組みあわせで絵をかこう

まだMSX2が出ていないか、出ていても高級機といわれていたころ、パソコンユーザーのあいだでCGがはやったことがある。CGといっても、CGツールを使ってかいた絵ではなく、プログラムでかく絵だった。

そうしたCGプログラムがさかんに雑誌に投稿され、たくさん掲載された。そのプログラムを打ちこむと自分のパソコンのディスプレイにあざやかなCGが再現されたのである。貧弱なたとえだが、大型のジグソーパズルが完成したときのような喜びに似ているのではないか。

なぜ最近そういう遊びがすたれてしまったのか。使いやすいCGツールがたくさん出まわってしまった結果、プログラムでCGをかくなんてめんどうくさい遊びは、なんとなくダサくなってしまったのかもしれない。

しかし、ファッションや歴史はくりかえすはずだ。掲載に値するようなCGプログラムがファンダムに投稿されたりする日も来るにちがいないと思いつつ、今回は、ドットの組みあわせでかく小さなCGの作り方をピクニックしてみよう。

ドットでかくのは、それがCGの基本だと思うから。小さなものにかぎったのは、単純に最初はデータ量の少ないものからはじめたほうがいいと思うから。

ここはMSX2/2+、SCREEN5でやってみよう。

たとえば下のような絵(25×25ドット)が右ページのような3ステップの手順と、かんたんな描画プログラムでできあがる。

↑右ページのデータの絵を1ドットずつかきこんでいるところ

↑大きさは25×25ドットなのでそうとう小さい(約9文字ぶんの大きさ)

たとえば25×25のドット絵

Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	1	1	1	1	1	Ø	Ø	1	Ø	Ø	1	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	1	1	1	7	7	7	1	1	1	1	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	1	1	1	7	7	7	7	7	1	7	7	1	1	1	1	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	1	1	7	7	7	7	7	7	7	7	7	1	7	7	1	1	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	1	1	7	7	7	7	7	7	7	7	7	7	7	7	1	7	7	1	1	Ø	Ø	Ø
Ø	Ø	1	1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	1	1	Ø	Ø
Ø	Ø	1	1	1	1	1	1	1	1	7	7	7	7	7	1	1	1	1	1	1	1	1	Ø	Ø
Ø	1	1	E	E	E	E	E	E	1	1	7	7	7	1	1	E	E	E	E	E	E	1	1	Ø
Ø	1	E	E	1	1	1	E	E	E	1	1	1	1	1	E	E	E	1	1	1	E	E	1	Ø
Ø	1	E	1	E	E	E	1	E	E	1	1	7	1	1	E	E	1	E	E	E	1	E	1	Ø
Ø	1	E	1	E	1	E	1	E	E	1	7	7	7	1	E	E	1	E	1	E	1	E	1	Ø
Ø	1	E	E	1	1	E	1	E	1	7	7	7	7	7	1	E	E	1	1	E	E	E	1	Ø
Ø	1	E	E	E	E	E	E	1	1	7	7	1	7	7	1	1	E	E	E	E	E	E	1	Ø
Ø	Ø	1	E	E	E	E	E	1	7	7	1	1	1	7	7	1	E	E	E	E	E	1	Ø	Ø
Ø	Ø	Ø	1	1	1	1	1	7	7	1	1	1	1	1	7	7	1	1	1	1	1	Ø	Ø	Ø
Ø	Ø	Ø	1	7	7	7	7	7	1	1	1	1	1	1	1	7	7	7	7	7	1	Ø	Ø	Ø
Ø	Ø	Ø	Ø	1	7	7	7	1	1	6	6	1	6	6	1	1	7	7	7	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	1	7	7	7	1	6	6	6	6	6	1	7	7	7	1	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	Ø	1	7	7	7	1	6	6	6	1	7	7	7	1	Ø	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	7	7	7	1	1	1	7	7	7	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	Ø	1	7	7	7	7	7	1	7	7	7	7	7	1	Ø	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	1	7	7	7	7	7	7	7	7	7	7	7	7	7	1	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	1	1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	1	1	Ø	Ø	Ø
Ø	Ø	1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	1	Ø	Ø

③DATA文にする

②色をコードに変える

①まず、方眼紙に絵をかく

リスト1：顔のデータ

```
1000 DATA ØØØØØØØØØØØØØØØØ1ØØ1ØØØØØ
1010 DATA ØØØØØØØØØØ111111ØØ1ØØ1ØØØ
1020 DATA ØØØØØØØ1111777111ØØ1ØØØØ
1030 DATA ØØØØØ11177777177111 1ØØØØØ
1040 DATA ØØØØ117777777771771110ØØØ
1050 DATA ØØØ117777777777771771 1ØØØ
1060 DATA ØØ117777777777777777771 1ØØ
1070 DATA ØØ111111117777711111111ØØ
1080 DATA Ø11EEEEEE1177711EEEEEE11Ø
1090 DATA Ø1EE111EEE11111EEE111EE1Ø
1100 DATA Ø1E1EEE1EE11711EE1EEE1E1Ø
1110 DATA Ø1E1E1E1EE17771EE1E1E1E1Ø
1120 DATA Ø1EE11E1E1777771EE11EEE1Ø
1130 DATA Ø1EEEEEE11771771 1EEEEEE1Ø
1140 DATA ØØ1EEEEE17711177 1EEEEE1ØØ
1150 DATA ØØØ1111177111117711111ØØØ
1160 DATA ØØØ177777111111177777 1ØØØ
1170 DATA ØØØØ17771166166117771ØØØØ
1180 DATA ØØØØØ1777166666177710ØØØØ
1190 DATA ØØØØØØ17771666177710ØØØØØ
1200 DATA ØØØØØØØ17771117771ØØØØØØØ
1210 DATA ØØØØØØ1777771777771ØØØØØØ
1220 DATA ØØØØØ177777777777771ØØØØØ
1230 DATA ØØØ1177777777777777711ØØØ
1240 DATA ØØ17777777777777777777 1ØØ
```

リスト2：データでドット絵をかく

```
10 COLOR 15,5,12:SCREEN 5,Ø
20 DEFINT A-Z:DIM A(158)
30 FOR Y=Ø TO 24:READ A$
40 FOR X=Ø TO 24:C=VAL("&H"+MID$(A$,X+1,
1)):PSET (X,Y),C:NEXT:NEXT
50 GOTO 5Ø
```

SCREEN5の画面は、ふつう横256×縦212のドットで構成されている。つまり、256×212の方眼紙だと思えばいい。

その方眼紙のマス目に色をぬっていくのが、ドット絵の基本なのだ。でも、いきなり画面全体にドット絵をかくのはたいへんだし、記事のうえでもつらいので、ここでは約87分の1にあたる25×25ドットの絵にしてみた。じつは前号に使った顔のキャラクタデータ(24×24)を修正していたら縦横に1ドットぶんのびてしまったのだ。

それはともかく、SCREEN5のマス目にぬることのできる色は基本的にパレットコード(カラーコード)1～15の15色プラス透明(パレットコード0)の16とおりだ。16とおりということは、そのデータは16進数1桁で表現することができるわけだ。1ドットの色のデータは16進数1桁、つまり0～Fの1文字に置き換えることができることになる。

①まず、方眼紙に絵をかく

方眼紙に15色プラス透明色の制約を守ってとりあえずドット絵をかいてみる。なれればデータで直接かけるだろうが、はじめはともかくタイルをならべている感覚でやってみよう。ちなみに、用意されているMSXの15色では満足できなくても、カラーパレット変更機能で512色のなかから自由に選べるので色のことはあとから考えてもいい。

②色をコードに変える

とにかく、なんらかの絵に見えるように方眼紙を埋めてみる。埋まったら、1色に1つのデータを割りふっていく。すると、25×25のデータ表ができあがる。これが絵のデータだ。

③DATA文にする

この絵のデータを、そのままここでは行1000からはじまるDATA文に移しかえる。1段ずつふつうの文字列としてあつかえばいいのだ。すると、リスト1のような顔のデータができあがる。

このデータとリスト2のような描画プログラムをあわせて実行すると26ページの画面になる。

なお、リスト2の行20はつぎのCOPY文のための配列宣言だ。

配列とCOPY　画面上のグラフィックを複写する配列は、倍精度型や単精度型でもいいのだが、ここでは整数型だけにかぎって説明する。また、使用するＳＣＲＥＥＮモードによっても変わる(ただしＳＣＲＥＥＮ５以上)が、ここではＳＣＲＥＥＮ５にかぎる。つまり、この記事で使用している場合にかぎった。この記事の場合は、ＤＥＦ ＩＮＴによって、配列Ａも整数型になっている。
配列にＣＯＰＹするグラフィックの横のドット数をＸ、縦のドット数をＹとすると、ＤＩＭ文で宣言すべき配列の添字の大きさは、
Ｘ＊Ｙ／４＋１
になる。この計算をおこなって端数が出た場合は切上げにする。

上下左右反転機能　あらかじめ配列にＣＯＰＹしておいた図形を画面にＣＯＰＹする場合の「方向」というパラメータはその図形を上下左右に反転させる機能がある。もとの図形は左上から右下に向かう方向で配列に入っているが、方向のパラメータに応じて次のような方向で画面上に表示される。
０＝そのまま(省略もおなじ)
１＝右上から左下(左右反転)
２＝左下から右上(上下反転)
３＝右下から左上(点対称)
これも使いようによってはアニメ効果などに便利だろう。

ロジカルオペレーション　ＳＣＲＥＥＮ５～７では、カラーコードが４ビット単位で論理演算されるが、ＳＣＲＥＥＮ８のみ、８ビット単位で論理演算される。これはＳＣＲＥＥＮ８のカラーコードが０～255と８ビットデータになっているため。また、通常の論理演算は16ビット単位でおこなわれているので、あらかじめカラーコードの論理演算の結果を予測するには、
Ｃ＝(Ａ〈論理演算子〉Ｂ)ＡＮＤ１５
として「ＡＮＤ１５」(または２５５)をつけた形で計算しないとわからなくなる。

配列に絵のデータが入る

リスト１＋リスト2のプログラムを走らせると、26ページの写真のように顔をかくわけだが、これを出発点とすると、2つの道への発展がある。

１つは、データを大きくしてもっと大きいドット絵をかくこと。これはリスト１、リスト2の仕組みさえわかればかんたんだ。たとえば、100×100の絵にしたいのならば、各行が100文字でできているデータを100行ぶん作り、リスト2の行30、40のループの終値をそれぞれ99に変えればいい。もっとも100×100のドット絵とデータ化はそれだけでかなりたいへんだが。

ただし、この調子でデータを増やしていってもSCREEN5の１画面全体を埋めるCGはかけない。メモリがたりないからだ。そこで、おなじ内容をもっと短い形のデータに置き換える方法(データ圧縮、などという)をいろいろ考える必要があるのだが、それはこっちにおいといて今回は素通り。

別の道は、いまかいた絵をモチーフにして画面を再構成する道だ。とりあえず、こっちのほうが楽ができそうなので、少しふみいってみよう。

■画面から配列へ

COPY(x1,y1)−(x2,y2)〔,〈ページ〉〕TO〈配列〉

■配列から画面へ

COPY〈配列〉〔,〈方向〉〕TO(x1,y1)〔,〈ページ〉,〈ロジカルオペレーション〉〕

【画面から配列へ】あらかじめ必要な大きさの配列を宣言しておいて画面に表示されているグラフィックを配列に複写。**【配列から画面へ】**配列の直後にある「方向」というパラメータと最後にある「ロジカルオペレーション」がおもしろい。

■配列に絵を複写する

SCREEN5というと、ゲームプログラムなどでは複数のページ(SETPAGEで指定する)とCOPY文で画面全体や絵柄の動きなどを表現することがひじょうに多い。その手法もここでは説明しないが、ファンダムのゲームプログラムやAVフォーラムの作品にいい例がたくさんあるので探してみよう。

あまり使われていないのが、配列に絵のデータをCOPYする方法だ。絵のデータを配列にCOPYするのは上の書式どおりだが、COPYする絵の大きさに応じた配列をあらかじめ宣言しておかなくてはいけない。絵の大きさと必要な配列宣言の公式は左の傍注にしるす。

配列に入れておいた絵のデータは、まるでパターン定義したキャラクタのように自由になんどでも使えるし、また表示スピードも速い。しかも、必要に応じて上下左右に反転して表示することもできるので、なにかいたずらができそうだ。

また、右ページでやっているロジカルオペレーションと組みあわせれば、思わぬグラフィック効果を楽しめるだろう。

もっとも、配列COPYで１つのキャラクタを動かす場合は、表示と同時に消すくふうも必要になり、スプライトよりは処理が複雑になるだろうが。

1つだけ顔がちがいます

1つだけちがう顔があります。3つ数えるあいだに探してください。1 2 3、ブー

リスト3：顔のCOPYと配列

```
50 COPY (0,0)-(24,24) TO A
60 CLS:C=0:R=RND(1)*81
70 FOR X=3 TO 252 STEP 25:FOR Y=6 TO 181
 STEP 25:C=C+1:H=-(C=R):COPY A,H TO (X+H
*24,Y):NEXT:NEXT
80 FOR I=0 TO 1:I=-STRIG(0):NEXT:GOTO 60
```

26ページのリスト１＋リスト2に、さらにリスト3をつないで実行してみると、まずゆっくり顔をかいて、それから左の画面のように80個の顔をすばやく表示する。この80個の顔のなかに１つだけほかとはちがった顔があるのでヒマな人は探してみよう。スペースキーを押せば、ふたたび80個の顔を表示して、またどこか別の場所に１つだけちがった顔をさりげなく置く。

プログラムは行50に注目。もともとリスト2の行20の配列宣言は、この行50で絵のデータを配列にCOPYするためのものだった。

こうして配列A(整数型になっている)にデータをCOPYしておけば、以後おなじ絵をすばやくくりかえして表示できるわけだ。しかも、上下左右の反転機能付き。

行70はおもに80個の顔を表示していく部分。変数Hはいつもは0だが、カウンタCが行20で決められた乱数Rになったときだけ、Hに1が入る。するとそのときだけここのCOPY文は右上から左下にかいていくことになる。つまり、左右反転しているわけだ。ただし、同時に表示開始位置も25ドットずらしていることに注意。そうしないとかいていく方向が逆なので、きっちり80個ならんだようには見えないのだ。

BASIC テクニック

COPYとロジカルオペレーション

TPSET　PSET　TPRESET　PRESET　TXOR　XOR　TOR　OR　TAND　AND

↑4色の四角と顔とを10種類のロジカルオペレーションで重ねあわせてみた。色の選び方で結果がぜんぜん変わってくるのでいろいろやってみるとおもしろい

リスト4：ロジカルオペレーション博覧会

```
50 COPY (0,0)-(24,24) TO A
60 CLS:LINE (0,0)-(11,11),4,BF:LINE (13,
0)-(24,11),8,BF:LINE (13,13)-(24,24),3,B
F:LINE (0,13)-(11,24),15,BF
70 DIM B(158):COPY (0,0)-(24,24) TO B
80 OPEN "GRP:" AS #1
90 CLS:COPY B TO (100,100):COPY A TO (13
0,100)
100 COPY B TO (115,130):COPY A TO (115,1
30),,TPSET  :PSET STEP(30,0):PRINT #1,"T
PSET   ":GOSUB 300
110 COPY B TO (115,130):COPY A TO (115,1
30),,TPRESET:PSET STEP(30,0):PRINT #1,"T
PRESET":GOSUB 300
120 COPY B TO (115,130):COPY A TO (115,1
30),,TXOR   :PSET STEP(30,0):PRINT #1,"T
XOR    ":GOSUB 300
130 COPY B TO (115,130):COPY A TO (115,1
30),,TOR    :PSET STEP(30,0):PRINT #1,"T
OR     ":GOSUB 300
140 COPY B TO (115,130):COPY A TO (115,1
30),,TAND   :PSET STEP(30,0):PRINT #1,"T
AND    ":GOSUB 300
150 COPY B TO (115,130):COPY A TO (115,1
30),, PSET   :PSET STEP(30,0):PRINT #1,"
PSET   ":GOSUB 300
160 COPY B TO (115,130):COPY A TO (115,1
30),, PRESET:PSET STEP(30,0):PRINT #1,"
PRESET":GOSUB 300
170 COPY B TO (115,130):COPY A TO (115,1
30),, XOR    :PSET STEP(30,0):PRINT #1,"
XOR    ":GOSUB 300
180 COPY B TO (115,130):COPY A TO (115,1
30),, OR     :PSET STEP(30,0):PRINT #1,"
OR     ":GOSUB 300
190 COPY B TO (115,130):COPY A TO (115,1
30),, AND    :PSET STEP(30,0):PRINT #1,"
AND    ":GOSUB 300
200 GOTO 100
300 FOR W=0 TO 3000:NEXT:RETURN
```

ロジカルオペレーションとは、直訳すると「論理演算」。基本的には、ＩＦ文のなかでよく見かける例のＡＮＤとかＯＲなどのことなのだが、ＳＣＲＥＥＮ５以上のグラフィック関係の命令（ＬＩＮＥ、ＰＳＥＴ、ＣＯＰＹなど）で使うロジカルオペレーションとは、色が重なったときの新しい色を計算するものなのだ。

画面上にもとあったカラーコードをＡ、新しく重ねられようとするカラーコードをＢとすると、ＡとＢで4ビット単位の論理演算をして、その結果の値が新しい部分のカラーコードとなるのだ。この最終的なカラーコードをＣとしておこう。

グラフィックのロジカルオペレーションには基本的に５種類あって、それぞれのＣ（最終的なカラーコード）は、Ａ、Ｂから次のように計算される。

①ＰＳＥＴ
Ｃ←Ｂ

つまり、ＢがそのままＣになる。これはロジカルオペレーションを省略した場合も同様で、ごくふつうに「上書き」する場合。命令のＰＳＥＴとおなじことばなので、ちょっととまどう。

②ＰＲＥＳＥＴ
Ｃ←ＮＯＴ　Ｂ

たとえばＢが青の４（＆Ｂ０１００）だとすると、ＮＯＴ　Ｂは＆Ｂ１０１１でカラーコード11の明るい黄色になる。

残りの3つのロジカルオペレーションは、Ａ、Ｂの2つのコードの値にそれぞれの論理演算をおこなって新しいカラーコードを計算する。それぞれ、Ａが５（明るい青）、Ｂが12（暗い緑）だとすると、

③ＸＯＲ
Ｃ←Ａ　ＸＯＲ　Ｂ
Ｃは９の明るい赤

④ＯＲ
Ｃ←Ａ　ＯＲ　Ｂ
Ｃは13の紫

⑤ＡＮＤ
Ｃ←Ａ　ＡＮＤ　Ｂ
Ｃは４の青

というふうになる。

そして、ＣＯＰＹだけは、この5種類のロジカルオペレーションの頭に「Ｔ」がつく５種類のロジカルオペレーションが追加される。Ｔはおそらくトレースの頭文字で、ＣＯＰＹした図形のうち透明色になっている部分だけはいかなる場合でももとにあった色が残るというもの。つまり、図形の透明色以外の部分を切り抜いて転送しているわけだ。これは、上のＴつきグループとＴなしグループを比べてみるとわかるだろう。顔のまわりは27ページを見ればわかるようにカラーコード０になっている。

リスト１＋リスト２に上のリスト４を追加して実行すると、このロジカルオペレーションのすべてを実演する（左上の10枚の写真）。つまり、４色にぬられた四角のうえに顔を10とおりのロジカルオペレーションで重ねてみたわけだ。

このプログラムは見た目に長いようだが、じつは、行100までと行200と行300を打ちこんでしまえば、あとは行100の行番号とロジカルオペレーション名の部分（２か所ある）を書きかえるだけでどんどんコピーできて、じっさいには打ちこむのがかんたんなのでぜひ試してほしい。

配列とCOPYで作る動くグラフィック

BASICテクニック

＃15 万華鏡と配列忍者

MSXのなかで万華鏡を作っていたある夜、顕微鏡や曼珠沙華や南無妙法蓮華経や阿鼻叫喚やその他もろもろの似たことばが極彩色で飛びまわった。

配列とグラフィック　リスト１の行50や行80のような書式で、グラフィックを配列におさめることができる。MSX2以上の機種でBASICマニュアルのCOPYの項目を見れば、くわしくのっているだろう。
配列宣言をするときの大きさは、コピーするグラフィックの大きさによってちがう。整数型配列の場合、おさめたいグラフィックの水平ドット数をX、垂直ドット数をYとすると、ＳＣＲＥＥＮ５の場合は、
Ｘ＊Ｙ／４＋２
よりも大きい配列を宣言する必要がある。リスト１の行20参照。
TXOR　COPY文で使うロジカルオペレーションの１つで、「Ｔ」は、コピーするグラフィックのうちパレットコード０の部分はまったく転送しないことを意味し、残りの「XOR」はコピー先とコピーするグラフィックの２つのパレットコードをXORで論理演算して決定するという意味だ。ただし、通常の論理演算とはちがい、２進数４桁で演算する。
裏画面　リスト１のプログラムでは裏画面となるページ１をCLSしていないので、いろいろ試しているうちにページ１がよごれて絵が変になることもあるだろう。ページ１をきれいにしたいときは、
ＳＣＲＥＥＮ５：ＳＥＴＰＡＧＥ１，１：ＣＬＳ
を直接実行すればいい。

COPY文で万華鏡を作ろう

職業が幼稚園児だったころ、わたしは万華鏡が大好きだった。サラサラときれいな模様が変化していくのがおもしろくてしかたがなく、よく万華鏡を見ながら道を歩いては、道路の穴に足をつっこんで泣いた。

その後、経験値をかせいでレベル3の小学生になったころ、万華鏡が3面のチャチな鏡と安っぽい色のついたプラスチック片でできていることを知って、マジックポイントが一気に減ったのを覚えている。それまで、万華鏡の内部に外の世界とは異なった別の世界があるのだと、ぼんやり思っていたからだ。

現代日本で忍者になるのはむずかしいと気づいたのもそのころだったような気がする。それまでなりたい職業は忍者だった。

ついでに思い出すと、アメリカには日本語のうまい人たちがいて、その人たちが日本向けのテレビ映画を作っているのだと思っていた。

そんなことを思い出しながらSCREEN5の画面上で配列をいろいろCOPYしたりしていじっていると、万華鏡もどきができた。右ページのリスト1がそうだ。

↑リスト１を実行すると画面中央でパタパタと万華鏡もどきがうごめく

ほんものの万華鏡は三角柱だし、できる映像は無限だが、とりあえずMSXのなかにできたのは四角くて有限なやつだった。

行70のTXORのせいで、思いがけない色の変化が次々に起こる。ずっと見ているとくりかえしになるのだけれど。

万華鏡とその舞台裏

リスト1:スクエアな万華鏡

```
10 COLOR 15,0,0:SCREEN 5:DEFINT A-Z
20 S=17:L=S*2:P=S:PP=1:DS=(S+1)^2/4+2:DL
=(L+1)^2/4+2:DIM A(DL),B(DS)
30 FOR I=0 TO S STEP 2:C=RND(1)*13+3
40 CIRCLE (S,S),I,C:NEXT
50 COPY (0,0)-(L,L) TO A:CLS
60 COPY A TO (0,0),1
70 COPY A TO (P,P),1,TXOR
80 COPY (P,P)-STEP(S,S),1 TO B
90 FOR I=0 TO 3:COPY B,I TO (130,100):NE
XT
100 P=P+PP:IF P>L OR P<3 THEN PP=-PP
110 GOTO 60
```

↑変化していく万華鏡のグラフィックから、あるときの変化のようすを順に追ってみた。いちばん上からいちばん右の写真まで1秒もない

リスト1でやっている作業の流れを箇条書きにすると、
①ランダムな色で同心円をかく＝行30〜40
②それを配列Aにコピー＝行50
③配列Aを裏画面(ページ1)に2つずらしてかきこむ＝行60、70
④できた模様を一部分切り取って配列Bにコピー＝行80
⑤配列Bを表画面に方向を変えて4つ表示する＝行90
⑥ずらし方(③)や切り取る場所(④)を変えながらくりかえす＝行100、110

■万華鏡効果は行90

配列Bにコピーされた裏画面のグラフィックが表画面で万華鏡ふうになるのは、⑤つまり行90で、点(130,100)を中心にしておなじグラフィックデータ(配列Bの中身)を対称に展開しているからだ。上の6枚の写真のそれぞれで四角いグラフィックを4分割してみよう。4分割したときの右下の図形、それが配列Bをすなおに単体で表示したときの姿なのだ。

行90のIのループの終値を0にして(つまりＦＯＲ Ｉ＝Ø ＴＯ Ø)リスト1を実行してみると配列B単体の姿がよくわかる。

変化するグラフィックをこのように4方向に対称に表示すれば万華鏡効果はかんたんに得られる。

■裏画面のグラフィック

配列B単体の変化も見ていると不思議な気がするが、この不思議さは、行70のＴＸＯＲのせいだろう。この、長い名前を持つロジカルオペレーションは、すでに表示されている図形の上に同心円グラフィック(配列A)をＸＯＲで重ねていく。「T」は配列Aのパレットコード0(透明)の部分は無視するという意味だ。

裏画面で起きていることをのぞいてみよう。次の行55を加えればのぞくことができる。

```
55 SETPAGE 1
```

これを加えてリスト1を実行すると、最初にかいた同心円のグラフィックが奇妙に重なって動いているのがわかる。その動きの一部が配列Bの変化になっているわけだ。

ほかにも行60の最後に「，ＴＰＳＥＴ」を加えてみたりなどいろいろ手を加えてみよう。

↑裏画面ではこんなふうにけっこう汚いグラフィックをかいている

配列の添字　たとえばA(n)という配列があるとすると、nが添字。なんの宣言もせずに配列を使うと、自動的に最大添字が10の配列を宣言したのとおなじことになる。また、11以上の添字を持つ配列を使いたいときは、DIM文で宣言しておく必要がある。
RAMの領域　配列はその配列全体で必要な量をまとめて確保される。グラフィック配列は、そのグラフィックが使用しているVRAMの容量に匹敵するメモリ領域を確保するためのもの。ところで、グラフィック配列が整数型だろうと、倍精度型だろうと、グラフィックデータのRAMへの収納のされ方は変わらない。頭の2バイトがX方向のドット数、次の2バイトがY方向のドット数、その次からグラフィック本体のデータが入る。しかし、整数型と倍精度型では要素変数1つが占めるメモリ領域がちがう。整数型の場合は、1つの要素変数に対して2バイトが割りあてられるので、A(0)がX方向のドット数、A(1)がY方向のドット数に対応するのだ。
リスト1～3では、すべてDEFINTで整数型宣言をしているので配列名になにも型宣言文字をつけていないが、整数型宣言をしていない場合は、たとえば、
DIM　A%(D)
のように%のついた配列名を使えばいい。

グラフィック配列の中身

ロジカルオペレーションによる色の変化もけっこう不思議だが、配列にグラフィックをコピーしたり、配列を画面にコピーしてグラフィックを表示したりしていること自体、かなり不思議なことだ。

グラフィックデータの入れ物として使う配列を、ここではグラフィック配列と呼ぶことにしよう。グラフィック配列はどういう仕組みになっているのか。

■領域の名前でしかない

そもそも配列変数というもの自体、ちょっと不思議なものだ。配列変数の1つ1つの要素はそれぞれ独立した変数のように扱える一方で、添字という背番号によって配列として団結している。メンバー全員がソロプレイヤーとして活躍するバンドのようなものだ。

しかし、グラフィック配列の場合は、ふつうの配列変数とはまったく使い方がちがう。基本的に配列全体を1つのものとしてしか使用しないのだ。

グラフィック配列は、その配列のためにRAM上に確保されたメモリ領域を利用しているだけなのだ。配列名は、たんにRAMの領域につけられた名前にすぎない。その領域と、VRAMの領域とでデータのやりとりをするのが、グラフィック配列のじっさいの使われ方なのだ。

■整数型配列の場合の利点

ところが、整数型のグラフィック配列の場合にかぎって、グラフィック配列の最初の2つの要素変数(添字が0と1のもの)の内容が、グラフィック自体のサイズに対応する。たとえばリスト2でいえば、グラフィック配列Aの先頭、つまり、A(0)にはそのグラフィックの水平ドット数、A(1)には垂直ドット数が入っているのだ。リスト2を実行したあとに、この2つの要素変数の内容をPRINTすれば確認できる。

↑リスト2の実行画面。変化するグラフィックを次々に重ねてコピーしているので、中央付近であやしいシマ模様がぐるぐる回転しながら現れては消えていく

で、話はここからだ。このA(0)、A(1)の値を変更すると、グラフィック配列によって表示するグラフィックそのものに変化がおきるのだ(とうぜんだが)。

リスト3は、その変化をはっきり目で確かめるために作ったものだ。まるで忍者のように変化しているのでかってに配列忍者と名づけた。

リスト2:配列忍者による万華鏡もどき

```
10 COLOR 15,0,0:SCREEN 5:DEFINT A-Z
20 S=19:R=(S+1)¥2
30 D=(S+1)^2/4+2:DIM A(D)
40 FOR Y=0 TO S:FOR X=0 TO S
50 IF (X-R)^2 + (Y-R)^2 > R^2 THEN 80
60 C=RND(-X-Y*S)*16
70 PSET (X,Y),C
80 NEXT:NEXT
90 COPY (0,0)-(S,S) TO A:CLS
100 FOR I=S TO 0 STEP -1 :A(0)=I+1
110 FOR J=0 TO 3:COPY A,J TO (130,100):N
EXT
120 NEXT
130 GOTO 100
```

配列忍者を作った動き

リスト３では、配列忍者の消え方がよくわかるように、まえに表示していたグラフィックをいちいち消しながら(行130で空のグラフィック配列をコピー)、新しいグラフィックを表示しているので、配列忍者の変化の仕方がよくわかるはずだ。

グラフィックをザーッと水平方向に消していき、そのあとふたたび逆まわしでグラフィックをじょじょに表示してくる。垂直方向についてもおなじ手順だ。

リスト３の行150の

A(H)=I

でグラフィック配列の構造を変えている。Hは、Wのループが終わるたびに０と１のどちらかに切り換えられる仕組みだ。また、そのすぐ下でXとYを入れ換えたり、Zに−１を与えたりしているのは、配列忍者の変化する方向を切り換えるためだ。

リスト２、３に共通の行50の条件式は、もとになるグラフィックをかくときに半径Rの円より外にあるドットはかかず、Rより内側にあるドットのみをかくようにするためのもの。

グラフィックの動きに見あきたら、リスト１〜３すべてについていえるが、配列を画面にコピーしている部分(リスト３でいえば行130)で、「,, TXOR」とか「,, TPSET」などのロジカルオペレーションを加えたりして改造を試してみよう。

リスト３：水平忍者と垂直忍者

```
10 COLOR 15,0,0:SCREEN 5:DEFINT A-Z
20 S=30:R=(S+1)¥2
30 D=(S+1)^2/4+2:DIM A(D),B(D)
40 FOR Y=0 TO S:FOR X=0 TO S
50 IF (X-R)^2 + (Y-R)^2 > R^2 THEN 70
60 PSET (X,Y),RND(-X-Y*S)*16
70 NEXT:NEXT
80 COPY (0,0)-(S,S) TO A:CLS
90 FOR I=0 TO 1:B(I)=A(I):NEXT
100 X=A(H):Y=1:Z=-1
110 FOR W=0 TO 1
120 FOR I=X TO Y STEP Z
130 COPY B TO (130,100)
140 COPY A TO (130,100)
150 A(H)=I:NEXT
160 SWAP X,Y:Z=-Z:NEXT
170 H=-(H=0):GOTO 100
```

A(Ø)…X方向のドット数
A(1)…Y方向のドット数

■グラフィック配列の先頭が持つ情報

水平忍者

↑X方向のドット数＝A(０)を小さくしていくと、ちょうどテキスト画面でWIDTHを変えたときのようにグラフィックが変化していく

垂直忍者

↑Y方向のドット数＝A(１)を小さくしていった場合は結果が明白でどんどん下のほうから消えていく

ファンダムパーラー 号外

今月は、「ファンダムパーラー」、「ちえ熱あっちゃん」、「ダンジョンRPGの過程」をお休みします。これというのも、『WHIZだもんね♥』をはじめとして、掲載プログラムが内容、ボリュームともに充実しているためですが、3コーナーがいっぺんにお休みではさみしいので、1ページだけいただいて、なにやら書いてあります。

今回、こんなにもプログラムが充実していたのは、ひさしぶりにFP部門での採用作品があったからといっても過言ではないでしょう。FP部門にはあの『まものクエスト』の作者TPM.COからも、パズル感覚のRPG『ネイティガの悪魔』が届いていましたが、こちらのほうは現在検討中です。

『WHIZ〜』の掲載で、第1回FP部門季間奨励賞の候補作品が、1本ノミネートされたことになりました。

来月のパーラーは、静かなブームを呼びつつある1行プログラム特集の予定です。

ちえ熱あっちゃん

落書　あっちゃん、考えこむ

こんにちは、たびかさなるちえ熱のため、休養をとることになった「あっちゃん」です。

それにしても、きました、きました！　送られてきました！「ちえ熱あっちゃんがんばって」と応援のハガキが。いや〜うれしいなあ。1枚もこなかったらどうしようと心配していたので感謝、感謝。

ほかにも、いろいろ送られてきた「ちえ熱あっちゃん」あてのメッセージを紹介すると、「人工知能みたいのをやってください」(無理だって)、「BASICをマスターしてない私には役に立つ」(ほんとうかなあ〜)、「自分とおなじぐらいのプログラムを作れる人がいてうれしい」、「このスタッフがどこまで成長するか興味がある」(大きなお世話だ)、「あっちゃんのイラストがかわいい」(なんだそりゃ)etc.……。

数少ない、メッセージを送ってくれた人、ありがとう。今度はこんなことをしてほしいみたいなことを書いて送ってきてよ。

これからの「ちえ熱あっちゃん」の展開としては、まずは今やっている「英単語カード」プログラムをもっと使えるものにするのが目的である。

どんなふうにするかというと、出題する問題を変えるためにわざわざプログラム中のデータを変えなくてもすむように、最初に「モンダイ？」、「カイトウ？」とコンピュータが聞いてきて、自分で自由に出題する問題を作れるようにしたいし、自分で問題を作るときに動詞の問題集、名詞の問題集、各レッスンごとの問題集などと目的に応じた問題集を作っていき、問題集ごとに別々のファイルにしてディスクにセーブできるようにしたい。勉強するときは、勉強したい問題集をロードするだけでわざわざ打ちこまなくてもすむからだ。そしてこれらのことをするためにディスクのファイル操作を学ぶことになるだろう(といいつつも、そこにどういう世界が待ちうけているのかまだ知らない)。

「英単語カード」が完成したら、違う教科の学習プログラム作りに移ることになるだろう。理科の学習プログラムなら、ある実験をシミュレートしてみたり、数学ならグラフをかいてみたり、へんてこな図形を描いて面積を求めてみたり、社会だと年号とそのときの出来事を「英単語カード」みたいに、1問1答形式でやってみるか、空白のある年表を示しその空白を埋めていく問題みたいのもいい。

とまあ、夢は広がっていくが、とりあえずは、まだ「英単語カード」用の問題集作成プログラムを作っていかなくては……。

【担当デスクより】まだ、プログラミングの入口に立ったばかりのあっちゃんは、やりたいことと自分にできることの区別がまだついていないようです。理科の実験をシミュレートするようなプログラムがほんとうにできたら、私はあっちゃんの忠実なマネージャーになってもいい。

ダンジョンRPGの過程

番外編　「ダンジョンRPGの過程」の過程

ダンジョンRPGというプログラムは、いったいどんなものになるのだろう。担当にも、まだ予測がつかない。

読者のみんなにこのページはどんなものなのかと、今の段階までの担当の苦労話をしたいとおもう。

この企画の主旨は、ファンダムに掲載された1画面の迷路ゲームをもとに、ダンジョンRPGを作る過程を報告することで、プログラムを作成する模範演技のようなページにするハズだった。つまり「ちえ熱あっちゃん」などより少しプログラムがわかる読者を対象にしているページなわけだ。ところが担当が欲ばりなせいか、プログラムはもとの1画面からN画面、10画面へと、どんどん長くなっていった。「これで模範演技？」とおもう読者もいるだろうが、プログラムを作っていく過程では、こういうことはよくあることで、生の報告だからこそ、長くなってしまったのだ。プログラムをはじめから効率よく作るには、かなりの経験が必要になってくるし、そういうプログラムは、スパゲッティのようにこんがらかっていることが多いので理解しにくく、そこからの展開もあまり望めないことが多い。だからはじめのうちはリストの長さにとらわれずに、すなおにプログラムを作っていくほうがいい。はじめから力まかせに作っていくと、根深いバグに悩まされるハメにもなりかねない。

担当が考えるすなおなプログラムというのは、初期設定やサブルーチン、データなどがきちんとまとまっているプログラムだ。第1回の「ダンジョンRPGの過程」や「ちえ熱あっちゃん」でフローチャートを載せたのは、こういった処理ごとにまとまりのあるプログラム作りを、覚えてほしかったからだ。また、フローチャートを作っておくと、プログラム作成がおもったよりスムーズにいく。

「ダンジョンRPGの過程」に掲載されたプログラムを、ただゲームとして遊ぶだけでなく、どんどん改造してみてほしい。

目的のダンジョンRPGを作っていく段階で、担当がいまだにひっかかっているところがある。3Dダンジョン表示サブの処理がそれで、広間と格子廊を実現するために、約2画面もくってしまった。いくらわかりやすくといっても、これでは長すぎる。いくつか別の方法も試みたが、どれもスピードが遅くなってしまい、ゲームとしては大きなマイナスだ。これは他の処理にも影響していき、担当としては長さに見あった内容にせざるをえなくなっていった。

モンスターとの戦闘シーンは、そういう背景から考え出された。リストの長さ以上に、楽しめるものにしたいとおもう気持ちは、プログラムを組む人すべてにあるだろう。長くてもすなおなプログラムと、長いながらそれ以上に楽しめるプログラムの両方が、ダンジョンRPGにあることを目指して、今回の番外編を終えるとしよう。

かんたんな3Dグラフィックの試み

BASICピクニック

#16 SCREEN7とワイアフレーム

ディスプレイの輝く平面上に立体を表現してみよう。初歩的で原始的で単純なやり方だけれど、それでもなんとか立方体の存在を感じることができるのだ。

3D　three dimensions(3次元)の略。「さんでぃー」と読むことが多い。ディスプレイはどうあがいても平面(2次元)なので、そのディスプレイ上のグラフィックが「3Dグラフィック」という名前を持ったりするのは、変な気もするが、これは「疑似3Dグラフィック」の「疑似」を省略した形だと思えばいい。ただし、今回のBASICピクニックでとりあげた「投影図」のように、単純な3Dグラフィックのことをいうときに、とくに「疑似」の2文字を意識的につける場合もある。

ワイアフレーム　wire frame(針金の枠)。正式には、ワイアフレーム・モデルという。針金の枠で作った立体のように、線だけで表現する3Dグラフィックのこと。線だけのデータなのですべての辺が見える。ふつうは、ワイアフレームでも奥行きがあるのがふつうだが、今回あつかった奥行きのない投影図は、そのワイアフレームのなかでも、もっとも原始的なタイプだといえる。3Dグラフィックには、このワイアフレームのほかに

●サーフィス・モデル(surface model)＝ワイアフレームに面(サーフィス)の情報も加えたもの。向こう側に隠れる線は表示しないなどの処理(陰線処理)をする。ふつう3Dグラフィックというと、この手のものを指すことが多い

●ソリッド・モデル(solid model)＝立体の内部のデータも加えた3Dグラフィック

立体のにおいを持つ投影図

大介はさておき、次元ということばは、日常的によく使われるわりになんだか得体のしれないことばである。

目の前にある現実世界は、縦・横・高さ・時間の4次元(時空)とも考えられるが、時間をコントロールできない以上、やはり3次元と考えたほうが心になじむ。これに対して、ディスプレイのなかはどうあがいても2次元でしかない。

3次元上のものを2次元上で表現しようとすると、1次元を落としてしまうしかない。3Dグラフィックのテクニックというのは、「じょうずな1次元の落とし方」ということになるわけである。

そもそも、視覚は、3次元の世界から放たれた光を網膜という2次元の感覚器官に受けることで成り立っている。つまり、視覚そのものが3次元から2次元への次元落としで成り立つ感覚なのだ。ということは、「3次元の視覚」とディスプレイ上のグラフィックとは、本質的にはそんなに変わらないものだという気がする。

3次元の形を2次元上で表現する場合、おもに①見取図、②投影図、③展開図の3種類の方法がある。ほんとうは①の見取図式のグラフィックをやりたいところだが、ここでは②の投影図をやってみよう。投影図こそもっともかんたんに1次元を落とす方法で、疑似3Dグラフィックへの入口なのだ。

↑たんにZ軸の座標成分を無視しただけの投影図だが、立方体らしいことは伝えられる

歪んだ方眼紙　SCREEN7

正方形を表示したい

(50,50)　(150,50)
(50,150)　(150,150)

まともにやるとこうなる

なんとか正方形になった

⊖ＳＣＲＥＥＮ７でごくふつうに正方形を表示しようとすると左のようにものすごく縦長になってしまう。この画面モードはＸ軸方向が圧縮されている感じだ
⊖Ｘ軸方向だけ1.7倍してやると右の写真のようになんとか正方形に見える。小さくなるが、Ｙ軸方向を1.7分の１しても同様

とりあえず、線画でいろいろやるのだから、ＭＳＸ２(ＶＲＡＭ128K)でラインがもっともきれいに出るＳＣＲＥＥＮ７モードを使えるようにしよう。

ＳＣＲＥＥＮ７は、横512ドット×縦212ドットの解像度を持っていて、横方向がＳＣＲＥＥＮ５の倍になっている。今回のＢＡＳＩＣピクニックのプログラムにかぎっていえば、ＳＣＲＥＥＮ６(ＶＲＡＭ64Ｋの機種にも使用可能)でもいいのだが、これは色が不自由なのだ。

ＳＣＲＥＥＮ７で正方形をかこうとすると、最初はうまくいかない。なぜかというと、ＳＣＲＥＥＮ７のドットのならび方は、縦横の比率がかなりいびつなものになっているからだ。

ためしに、リスト１の行20の「：Ｐ＝１．７：Ｘ＝Ｘ＊Ｐ」を削除して、リスト１を実行してみよう。行30は、(50，50)を左上の頂点として、縦横101ドットの正方形をかく部分だが、結果は上の写真のように、縦長の妙な四角になってしまう。

この縦横比のずれを直すには２とおりの方法がある。１つはＸ座標成分を増幅する方法、もう１つはＹ座標成分を縮小する方法。ここでは、前者を選んだ。

このＳＣＲＥＥＮ７の縦横比を修正するための部分がさきほど削った「：Ｐ＝１．７：Ｘ＝Ｘ＊Ｐ」だったのだ。

(50，50)の点からＹ座標方向へ100ドット、Ｘ座標方向へ100×1.7つまり170ドットの四角をかくと、右上の写真のように正方形になる。このようにＸ成分だけを1.7倍して表示すればＳＣＲＥＥＮ７はノーマルな方眼紙としてとりあつかうことができる。

Ｐの値は、編集部で使っているディスプレイを見ながら決めたものなので、じっさいに使う場合は自分のディスプレイにあわせて微調整しておこう。

また、Ｙ座標の方向に縮小する方法をとりたい場合は、「Ｘ＝Ｘ＊Ｐ」のかわりに、「Ｙ＝Ｙ／Ｐ」を置けばいい。

リスト1：SCREEN7の正方形

```
10 COLOR 15,0,0:SCREEN 7
20 X=100:Y=100:P=1.7:X=X*P
30 LINE (50,50)-STEP(X,Y),15,B
40 GOTO 40
```

ワイアフレームに必要なデータ

■１辺101ドットの立方体の例

ここでは、ワイアフレームで立体を映すつもりなので、
①各頂点の３次元座標セット
②各辺の組み合わせデータセット
の２つが最低限必要になる。

①のためには座標軸を決めておかなくてはいけないが、左の図のように、Ｘ、Ｙ軸の方向をディスプレイにあわせてとっておいた。ふつう数学の教科書などで出てくるものとはちょっとちがうが、こういう座標系を使ったほうが思いえがいた像をデータ化するときにまぎれがなくていいと思う。

立方体の場合は、８つの頂点があるので、それに０～７の番号を振り、どの頂点と頂点が結ばれるかを番号の組み合わせで表現したのが、②のデータだ。

左のような立方体をワイアフレームで表現するために必要なデータを最初にプログラム化すると、①が行20～90の８セットのデータ、②が行110～220のように12セットのデータになる。

リスト2：立方体のデータ

```
10 '●●● POINT
20 DATA   0,  0,  0
30 DATA 100,  0,  0
40 DATA 100,  0,100
50 DATA   0,  0,100
60 DATA   0,100,  0
70 DATA 100,100,  0
80 DATA 100,100,100
90 DATA   0,100,100
100 '●●● LINE
110 DATA 0,1
120 DATA 1,2
130 DATA 2,3
140 DATA 3,0
150 DATA 4,5
160 DATA 5,6
170 DATA 6,7
180 DATA 7,4
190 DATA 0,4
200 DATA 1,5
210 DATA 2,6
220 DATA 3,7
```

ATN関数　リスト3の行1020にあるATN関数は、TAN関数の逆関数というやつで、

TAN(ATN(a))=a

という性質がある(じっさいにプログラムで試すと、左のaと右のaにわずかな誤差が出る)。ATN関数は、そのような値のうちで最小の値を返す関数なのだ。

さて、このaに1をあてはめると

TAN(ATN(1))=1

ところで、TAN(π/4)も1なので、ATN(1)は4分のπらしい。ということは、ATN(1)の4倍はπではないか、と思って

PRINT　ATN(1)*4

を実行すると、

3．141592……

と表示される。πの正確な値は、こんなところにあったのだ。

リスト3の行1020のAは、このATN(1)を利用して、36分のπラジアン(5度)に設定されている。

TAN(1)∧TAN(1)　リスト3の行1110の

W=TAN(1)∧TAN(1)

は計算結果自体は無意味だが、計算にかけている時間に意味がある。この計算をしている時間は、整数型のループ変数を使った場合の空ループ1200回ぶんくらいの待ち時間効果があるのだ。

三角関数は、このように待ち時間用に使われることがたまにあるが、TAN(1)はそのなかでも時間のかかる計算のようで、この場合は、さらに指数計算と組み合わせているのだ。

かんたんな1次元の落とし方

■投影図とは

夜、電灯など人工的な光源でできる影と、昼、太陽光線でできる影とはちょっとちがう。

電灯でできる影は、光源に近い頭の影は大きく、足に近いほうの影は小さく、影に奥行きが感じられる。しかし、太陽光線でできる影は、ただぺったりと地面と体のあいだに平行線を引いてかいたような影だ。

投影図の影は、後者の太陽光線でできる影とおなじように、奥行きのない影だ。だからこそ、データをいじるのが楽なのだ。

立体をディスプレイ平面上に投影した場合を考えてみよう。25ページの立方体のデータを作るときに使った座標系とおなじものをディスプレイ平面と組み合わせてみると、右上の図のようになる。X、Y軸はディスプレイ上のX、Y軸とおなじ。Z軸は、ディスプレイから垂直に出てくるものとする。

この場合、Z軸の方向と光の方向は完全に一致しているため、ある点のZ座標がなんであっても投影図にはなんの影響もないことがわかる。ということは、つまり、3次元座標系の座標データ(x, y, z)は、投影図の座標データとなるときに、単純にZ軸の座標を無視して(x, y)という形にすればいいわけだ。かくして、各軸に平行に作られた立方体は、図のようにただの正方形になってしまう。

しかし、これで終わっては人生の意味がない。

■ディスプレイと座標軸の関係

(0,0)　(100,0)　(0,100)　(100,100)　X　Y　Z　光の方向

■X軸を中心として回転すると

Z　c　b　(a,b,c)　X　Y　b sinθ+c cosθ　b cosθ−c sinθ　θ

■回転でよみがえるZ座標

各軸を中心に回転すると、その角度に応じて座標は27ページの下にある公式のように変わる。

たとえば、X軸を中心にして点(a, b, c)を角度θだけ回転させた場合、各座標は次のように変わる。

X←a

Y←b・cos θ−c・sin θ

Z←b・sin θ+c・cos θ

そして、この点をXY平面(ディスプレイ)に投影するために単純にZ座標を抜くと、

(a, b・cos θ−c・sin θ)

となる。ここに当初の3次元の座標データa、b、cの3つともが顔を出す。これこそ、投影図に立体のにおいを吹きつける大きなポイントなのだ。

以上のことをプログラムにすると、リスト3(ただしリスト2への追加)になる。このプログラムは単純な構造なので解説を見て流れを追ってみよう。

⬆リスト2+リスト3の実行画面。RUNした直後は、右上図のようにただの正方形を表示している

⬆とりあえず、Z軸を中心に回転させてみる。正方形が傾いたが、とうぜんまだ最初の立体のZ座標は顔を出さない

⬆そこで、Y軸中心に回転。すると、いままで重なっていた点や線が投影図に顔を出して立体らしくなってきた

3Dグラフィックへの入口

【リスト3の使い方】かならず25ページのリスト2(立方体のデータ)に追加してリスト3を打ちこむこと。カーソルキー4方向で次の操作をおこなう。①カーソルキーの上=回転方向の反転、②右=Y軸を中心とした回転、③下=X軸を中心とした回転、④左=Z軸を中心とした回転。なお、①は短い効果音、②〜④はそれぞれのテーマメロディが鳴る。

【解説】■1000〜1050●行1020は、変数の初期設定。P=縦横比率補正用/OX、OY=表示用の原点。すべての点に(OX, OY)を加えて表示している。この変数の値を変えると投影図の表示位置が全体にずれる/A=回転するときの角度。現在は5度(36分のπ)に設定。26ページ傍注参照。●行1030〜1040は、座標用配列変数の宣言/X(N)、Y(N)、Z(N)は頂点の3次元座標。XX(N)、YY(N)はディスプレイ平面上の頂点の座標(じっさいはほかの加工が加わる)。Nは頂点の識別番号の最大値/S(M)、G(M)は線で結ばれる頂点の組み合わせデータ。Mは辺の識別番号の最大値。●行1050は、PSG初期化とMMLの設定。配列A$(n)は、0から順に、回転角度反転の効果音その1、その2、Y軸回転の効果音、X軸回転の効果音、Z軸回転の効果音。■1060〜1080●行1040〜1060で宣言した配列にリスト2のデータを読みこむ。X(n)、Y(n)、Z(n)には各頂点の座標が入る。S(m)、G(m)に、識別番号mの辺の始点と終点になる頂点の識別番号が入る。●行1080は最初の表示のため。■1090〜1200●ここがキー入力処理や表示を含むメインルーチン。●行1100〜1110は、スティック入力の受け付けと処理。上が押されれば回転方向の反転処理もおこなう。変数Wはただの時間待ち用。26ページの傍注参照。●行1120は、変数Sを2で割って操作番号(1〜3が右、下、上に対応)に変え、効果音を出す。●行1130は、操作番号に応じて各サブへ。●行1140〜1160は、3次元座標から2次元座標(表示用)への変換。比率補正などを含む。●行1170〜1200は、各辺をかいてまた行1090へ。■各種サブ●行1210〜1380は、下の公式に基づく回転後の座標を計算するサブルーチン。それぞれの座標での計算結果を変数X、Y、Zに入れ、1回ごとに行1390〜1400のサブルーチンに行って新しいデータに入れ換える。

■回転後の座標

●X軸中心にAラジアン回転

X←X
Y←Y*COS(A)−Z*SIN(A)
Z←Y*SIN(A)+Z*COS(A)

●Y軸中心にAラジアン回転

X←Z*SIN(A)+X*COS(A)
Y←Y
Z←Z*COS(A)−X*SIN(A)

●Z軸中心にAラジアン回転

X←X*COS(A)−Y*SIN(A)
Y←X*SIN(A)+Y*COS(A)
Z←Z

リスト3:立方体のワイアフレーム

```
1000 '●●● SETTING
1010 COLOR 15,0,0:SCREEN 7
1020 P=1.7:OX=150:OY=50:A=ATN(1)/9
1030 N=7:DIM X(N),Y(N),Z(N),XX(N),YY(N)
1040 M=11:DIM S(M),G(M)
1050 PLAY"O5V15L8T150":A$(0)="BC":A$(1)=
"CB":A$(2)="L4DCDEGEDL8":A$(3)="G4EL3CEG
O6C4O5L8":A$(4)="CDE4CDE4GEDCDED"
1060 '●●● DATA READ
1070 FOR I=0 TO N:READ X(I),Y(I),Z(I):NE
XT:FOR I=0 TO M:READ S(I),G(I):NEXT
1080 GOTO 1140
1090 '●●● MAIN
1100 S=STICK(0)
1110 IF S=0 THEN 1100 ELSE IF S=1 THEN A
=-A:PLAY A$(-(A>0)):W=TAN(1)^TAN(1):GOTO
 1100
1120 S=S¥2:PLAY A$(S+1)
1130 ON S GOSUB 1270,1210,1330
1140 FOR I=0 TO N
1150 XX(I)=X(I)*P+OX:YY(I)=Y(I)+OY
1160 NEXT:CLS
1170 FOR I=0 TO M
1180 LINE (XX(S(I)),YY(S(I)))-(XX(G(I)),
YY(G(I)))
1190 NEXT
1200 GOTO 1090
1210 '●●● X-JIKU
1220 FOR I=0 TO N
1230 X=X(I)
1240 Y=Y(I)*COS(A)-Z(I)*SIN(A)
1250 Z=Y(I)*SIN(A)+Z(I)*COS(A)
1260 GOSUB 1390:NEXT:RETURN
1270 '●●● Y-JIKU
1280 FOR I=0 TO N
1290 Y=Y(I)
1300 Z=Z(I)*COS(A)-X(I)*SIN(A)
1310 X=Z(I)*SIN(A)+X(I)*COS(A)
1320 GOSUB 1390:NEXT:RETURN
1330 '●●● Z-JIKU
1340 FOR I=0 TO N
1350 Z=Z(I)
1360 X=X(I)*COS(A)-Y(I)*SIN(A)
1370 Y=X(I)*SIN(A)+Y(I)*COS(A)
1380 GOSUB 1390:NEXT:RETURN
1390 '●●● NEW DATA SUB
1400 X(I)=X:Y(I)=Y:Z(I)=Z:RETURN
```

もうちょっと進んだ3Dグラフィック

BASICテクニック

#17 まわるダイアモンド

前回は奥行き感のないワイアフレームを歩いてみた。意外とかんたんだった。今回はちょっと遠出して、ダイアモンドにおける奥行きの表現を走り抜けよう。

奥行きを感じさせるための2つの仕掛け

これは9月号のBASICピクニック#16からの続きだ。

■前回までのあらすじ

①3次元のものを2次元で表現するためには1次元ぶん落とす必要がある

②たとえば、Z軸がディスプレイ平面と垂直に交わることにしておけば、3次元座標空間の点(x,y,z)は、ディスプレイ上ではZ座標を落として、点(x,y)で表現できる

これが、ワイアフレーム式3Dグラフィックのなかでももっともかんたんな投影図の基本的な考え方だ。あまりに単純で、なんとなくごまかされたような気になる。じっさい、X、Y、Z軸に平行な辺でできた立方体をこの方式でディスプレイ上に表現するとただの正方形にしかならない。しかし、その立方体をいろんな方向に回転してみると、投影図はあるていどの立体っぽさを見せてくれる。

↑リスト1＋リスト3(32〜33ページ)を実行した画面。明るさとパースの立体感

3Dグラフィックのための最低限の知識は、Z座標を落とすことと、3次元座標空間のある点を回転したときの座標の変化の法則を知ることなのだ。

その法則には、もちろん公式がある。三角関数の加法定理を使えばかんたんに証明できる公式だが、証明はともかく、ある3次元の点(x,y,z)が、たとえば、X軸を中心にθ(シータ)ラジアン回転移動すると、新座標(x',y',z')は、次のような公式で計算される。

x'←x

y'←y＊cos(θ)−z＊sin(θ)

座標変換公式　Y軸とZ軸もついでに再掲しておこう。

・Y軸を中心にθ回転

x'←z＊sin(θ)＋x＊cos(θ)

y'←y

z'←z＊cos(θ)−x＊sin(θ)

・Z軸を中心にθ回転

x'←x＊cos(θ)−y＊sin(θ)

y'←x＊sin(θ)＋y＊cos(θ)

z'←z

じつはこの公式の証明方法を教えてほしいという質問がきていたのだが、そういうスペースはないので数学の先生にでもきいてみよう。

パース　パースペクティブ(perspective)の略。もろに「遠近画法(の)」という意味。

→

↑リスト２＋リスト３の実行画面（連続）。参考までに、ＡＩＷＳＸとＡＩＳＴ（turboR）の２機種でやってみた。30ステップぶんの時間は前者で111秒、後者で23秒だった

z'←y＊sin(θ)＋z＊cos(θ)

ここでディスプレイ上での変化を考えてみると、点(x,y)がＸ軸を中心にθラジアン回転すると、点(x,y＊cos(θ)−z＊sin(θ))に移動することになる。点の座標はどちらも単純にＺ座標の成分を落としているだけだが、移動後の点の座標には、もとの点のＺ座標の要素(z)がちゃんと顔を出している。つまりはじめの点の奥行きに関する情報(z)が、回転によってディスプレイに現れるのだ。

ｚに具体的な数値をあてはめて考えてみればわかりやすい。ｚが０のときと100のときを考えてみよう。このどちらでももとの点のディスプレイ上の表示は変わらないが、回転したときの移動先が大きく変わるのは明らかだ。その変わり方の違いがもとの点のＺ座標成分を間接的に表してくれるのだ。

■今回の新しい２つの要素

今回は、前回の投影図式３Ｄグラフィックに、２つの要素を加えてみた。１つは、透視図的要素、もう１つは明るさによる遠近感だ。

透視図的要素とは、いわゆるパースというやつで、手前ほど大きく、奥にあるほど小さく見せて遠近感を出す。原理はともかく、プログラム上の仕掛けは単純で、投影図へのちょっとした追加ていどになる。

後者の明るさによる遠近感とは、手前のものは明るく、遠くのものは暗く表示するというもの。もちろん、実際には光のあたり具合によって部分的な明るさは決まるわけで、遠くにある部分がかならずしも暗いわけではないのが現実だが、それ以上に人間の視覚は思いこみが激しく、明るいと手前だと思い、暗いと奥だと思うらしい。らしいというか、わたしはそうだ。

明るさのほうはプログラムで実現するのにちょっと苦労した。線が重なったときの処理、カラーパレット、何段階の明るさを使うか、明るさを決める基準をどこに置くか。くわしくは次のページで説明する。

前回の素朴な３Ｄグラフィックプログラムと、今回のやや進化したプログラムとの大きな違いはこの２点だ。

左ページの立方体の写真はどの辺が手前にあり、どの辺が奥にあるかがわかるはずだし、上の30枚の連続写真のようにクルクルと（実際にはク、ル、ク、ルだが）回転している物体もほとんどの人が「ダイアモンド」だと見てくれるはずだ。

リスト1、2のデータ構造　リスト1も2も共通する4つのREM文がある。
・N, M, OX, OY
Nは立体図形の頂点の数−1、Mは辺の数−1。OX、OYは表示するときの原点となる座標
・X(n)、Y(n), Z(n)
行ごと(つまり3つのデータごと)に各頂点のX、Y、Z座標。頂点には0からはじまる通し番号がついている
・A(n), B(n)
各辺に0からはじまる通し番号をつけてn番の辺の両端にある頂点の番号(上述の通し番号)。リスト3の行1320のLINE文で、n番の辺を描くとき始点となる頂点の番号がA(n)、終点となる頂点の番号がB(n)。データは、A(n)、B(n)の順に2つずつならぶ。なお、辺の通し番号の最大値は上述のM
・A：回転
リスト3の行1180で読みこんで、立体を回転させる軸を決めたり、逆回転させたりするデータ。それぞれ、0＝最初のデータからくりかえす、1＝X軸を中心に回転、2＝Z軸を中心に回転、3＝Y軸を中心に回転、4＝逆回転

遠近感のために具体的にどうするか

前回は立方体をカーソルキーで回転させるサンプルプログラムを掲載したが、今回はリスト1、リスト2の行510のデータにしたがって各軸を中心に回転するものを作ってみた。

リスト1＋リスト3で立方体の自動回転、リスト2＋リスト3でダイアモンドの自動回転のプログラムになる。リスト1とリスト2のデータ構造は左の傍注で解説するので、そちらを参照しててきとうにデータを変えて遊んでみてほしい。

リスト3を順に解説しよう。

■初期設定

1010　計算を多少でも速くするため。この行のあるのとないのとではMSX2(FS−5000)で時間を比べてみると30ステップ動くあいだに25秒の差が出た

1020　画面設定

1030　ページの設定。Pにディスプレイページ(見えるページ)、Qにアクティブページ(かきこむページ)の番号が入る。この行のSCREEN文はCLSでは消えないものを消すため、念のために入れたもの

1040　パレット設定

1050　画面の縦横比調整用

1060　XMはすべての頂点のX座標平均値。YM、ZMは同様にY座標、Z座標平均値。立体の中心座標。

1070・1080　Uは24分のπラジアン(7.5度)、立体が回転する単位角度。2次元配列T(n, m)は回転による座標値の変化を計算するための係数。どの軸で回転したときでもこの配列Tが使えるようにするためのもう1つの配列がC(n)

■データ読み出し

1100〜1150　リスト1やリスト2のデータを読み出す。1100ではN、Mに応じた配列宣言。1120ですべての座標をXM〜ZMに加え、1130でそれぞれ変数Iで割っているが、このIはループを終了したばかりなのでN＋1、すなわち頂点の数になっているのでそれを利用したもの。1140でXM〜ZMの存在理由が明らかになる。すべての頂点の座標からXM〜ZMを引くということは「原点」をちょうど立体の中心点に設定するということなのだ。これをしないと原点は立体の左肩(立方体でいえば左上奥の頂点)になり、振り回す感じの回転になる(前回の回転がそうだった)。1150は各辺がどの点とどの点を結ぶかのデータの読み出し

1160　回転方向のデータのある行500にデータ読みこみ先を設定して、1270からはじまる描画ルーチンへ飛ぶ(初期状態の描画)

■回転運動

1180〜1210　回転方向のデータから1つずつ読みこんで、Aが0ならまたはじめから4なら逆

ダイアモンドの頂点番号と最初の座標データ用の座標軸を表示してみたもの

リスト1：立方体用データ

```
10 '●●●  N,M,OX,OY
20 DATA  7,11,256,100
30 '●●●  X(n),Y(n),Z(n)
40 DATA    0,   0,   0
50 DATA  100,   0,   0
60 DATA  100,   0,100
70 DATA    0,   0,100
80 DATA    0,100,   0
90 DATA  100,100,   0
100 DATA 100,100,100
110 DATA   0,100,100
120 '●●● A(n),B(n)
130 DATA 0,1,1,2,2,3,3,0
140 DATA 4,5,5,6,6,7,7,4
150 DATA 0,4,1,5,2,6,3,7
500 '●●● A:ｶｲﾃﾝ
510 DATA 1,1,1,2,3,3,3,3,3,3,3,3,3,3,3,3
,3,3,3,3,3,3,3,3,3,3,3,3,3,3,4,3,3,3,3,2
,2,2,2,2,4,2,2,2,2,2,1,1,3,3,3,3,0
```

リスト2：ダイアモンド用データ

```
10 '●●●  N,M,OX,OY
20 DATA 12,29,256,80
30 '●●●  X(n),Y(n),Z(n)
40 DATA   44.282,  0      ,123.301
50 DATA   19.282,  0      , 80
60 DATA   44.282,  0      , 36.6988
70 DATA   94.282,  0      , 36.6988
80 DATA  119.282,  0      , 80
90 DATA   94.282,  0      ,123.301
100 DATA  69.282, 22.9825,160
110 DATA   0     , 22.9825,120
120 DATA   0     , 22.9825, 40
130 DATA  69.282, 22.9825,  0
140 DATA 138.564, 22.9825, 40
150 DATA 138.564, 22.9825,120
160 DATA  69.282,102.983 , 80
170 '●●● A(n),B(n)
180 DATA 0,1,1,2,2,3,3,4,4,5,5,0
190 DATA 6,7,7,8,8,9,9,10,10,11,11,6
200 DATA 0,6,0,7,1,7,1,8,2,8,2,9,3,9,3,1
0,4,10,4,11,5,11,5,6
210 DATA 6,12,7,12,8,12,9,12,10,12,11,12
500 '●●● A:ｶｲﾃﾝ
510 DATA 1,1,1,2,3,3,3,3,3,3,3,3,3,3,3,3
,3,3,3,3,3,3,3,3,3,3,3,3,3,3,4,3,3,3,3,2
,2,2,2,2,4,2,2,2,2,2,1,1,3,3,3,3,0
```

回転に設定(実際の回転はしない)。Aが1〜3なら1を引いて次の処理へ
1220〜1260　回転による座標の変化の計算。背後にある公式は前回とまったくおなじだが、X軸回転、Y軸回転、Z軸回転用の計算を1つにまとめた。配列C(n)はたんに0〜2を循環して使うためのものでじつはMODでもいい。この1230〜1250に順に出てくる係数のならび方はそれぞれX〜Z軸回転に応じて実際には下の内容一覧表に対応した数値になる
1270〜1290　ここで3次元データを2次元データ(描画用)にする。Wは奥行きで、500と600がパースのきつさを決めている。XX(n)、YY(n)が描画用の2次元データ
1300　ページ切り換え。この行で描画するページと見せているページをつねに別々に切り換えているためグラフィックの変化がスムーズに見える(おそいことはおそいが)。たんなるCLSだとチラチラして見にくい
1310〜1330　描画。Lは描画するときの線分のパレットコード。両端の点が原点より向こうにある程度が高いほど、15、7、3、1のうち小さいパレットコードになる。この数値はOR演算で重ねたときに都合のいいものを選んだ。それぞれのパレットの色(明るさ)は行1040で設定
1340　1170に飛ぶ

■配列Tの内容一覧表

X軸回転

1	0	0
0	0.991445	-0.130526
0	0.130526	0.991445

Y軸回転

0.991445	0	0.130526
0	1	0
-0.130526	0	0.991445

Z軸回転

0.991445	-0.130526	0
0.130526	0.991445	0
0	0	1

※数値は行1230〜1250で使われる配列Tの内容を各行ごとに順にならべたもの

リスト3：立体回転プログラム

```
1000 '●●● ショキ セッテイ
1010 DEFINT A-Q:DEFSNG R-Z
1020 COLOR 15,0,4:SCREEN 7
1030 P=0:Q=1:SETPAGE P,Q:SCREEN,0:CLS
1040 COLOR=(4,1,1,2):COLOR=(7,6,6,6):COL
OR=(3,5,5,5):COLOR=(1,3,3,3)
1050 R=1.7
1060 XM=0:YM=0:ZM=0
1070 U=ATN(1)/6:DIM T(2,2):T(0,0)=1:T(1,
1)=COS(U):T(2,2)=T(1,1):T(2,1)=SIN(U):T(
1,2)=-T(2,1)
1080 DIM C(5):FORI=0TO1:FORJ=0TO2:C(J+I*
3)=J:NEXT:NEXT
1090 '●●● デ゛ータ ヨミダ゛シ
1100 READ N,M:DIM X(N),Y(N),Z(N),XX(N),Y
Y(N):DIM A(M),B(M):READ OX,OY
1110 FOR I=0 TO N:READ X(I),Y(I),Z(I)
1120 XM=XM+X(I):YM=YM+Y(I):ZM=ZM+Z(I)
1130 NEXT:XM=XM/I:YM=YM/I:ZM=ZM/I
1140 FOR I=0 TO N:X(I)=X(I)-XM:Y(I)=Y(I)
-YM:Z(I)=Z(I)-ZM:NEXT
1150 FOR I=0 TO M:READ A(I),B(I):NEXT
1160 RESTORE 500:GOTO 1270
1170 '●●● ウコ゛ク
1180 READ A
1190 IF A=0 THEN RESTORE 500:GOTO 1180
1200 IF A=4 THEN SWAP T(2,1),T(1,2):GOTO
 1180
1210 A=A-1
1220 FOR I=0 TO N
1230 X=X(I)*T(C(A),C(A))+Y(I)*T(C(A),C(A
+1))+Z(I)*T(C(A),C(A+2))
1240 Y=X(I)*T(C(A+1),C(A))+Y(I)*T(C(A+1)
,C(A+1))+Z(I)*T(C(A+1),C(A+2))
1250 Z=X(I)*T(C(A+2),C(A))+Y(I)*T(C(A+2)
,C(A+1))+Z(I)*T(C(A+2),C(A+2))
1260 X(I)=X:Y(I)=Y:Z(I)=Z:NEXT
1270 FOR I=0 TO N:W=(Z(I)+500)/600
1280 XX(I)=X(I)*R*W+OX:YY(I)=Y(I)*W+OY
1290 NEXT
1300 SETPAGE P,Q:CLS:SWAP P,Q
1310 FOR I=0 TO M:L=2^(4+(Z(A(I))<0)+(Z(
B(I))<0)+(Z(A(I))+Z(B(I))<0))-1
1320 LINE (XX(A(I)),YY(A(I)))-(XX(B(I)),
YY(B(I))),L,,OR
1330 NEXT
1340 GOTO 1170
```

科学技術計算にチャレンジするMSX

BASICピクニック

#18 電子計算機としてのMSX

コンピュータはえらい。かつては人の一生を費やして計算したような問題をほいほいとやってのける。MSXもそんなコンピュータの一員なのだ。

MSXはどこまでコンピュータか？

今回のBASICピクニックのテーマは、MSXに科学技術計算をやらせてみよう、というものである。

このテーマは人によってバカバカしく聞こえるだろうし、そのバカバカしさも人によっては正反対のものだろう。1つは、「MSXで科学技術計算をやるなんてあたりまえじゃないか」というもので、もう1つは「MSXで科学技術計算なんかしてもしかたがない」というものだ。

完全に無関心な人はべつとして、もう一種類、「MSXに科学技術計算なんかできるの？」と感じる人々もいるだろう。

この記事は、2番目のタイプと3番目のタイプの人に向けて書かれている。1つのかんたんな事実を思い出すか、気がついてもらいたいからである。

その事実とは、MSXが電子計算機だということだ。

MSXは、BASICという科学技術計算に適した高級言語を内蔵するコンピュータなのだ。

その証拠に、たとえば、SIN、COSはともかく、ATNとかLOGとかEXPとか、小遣い(づか)やテストの成績の計算には不必要な、難解そうな関数がちゃんと用意してある。正直にいってEXPなんて、いったいどういうふうに役に立つのか、かくいうわたしにもよくわからないくらい科学技術なのだ。これで6個も科学技術ということばが出てきた。いまので7個目だ。

科学技術科学技術科学技術とくりかえしているうちに、頭がボーとしてきて、どういう意味だったのか思い出せなくなってしまいそうだが、コンピュータでやってみせる科学技術計算パフォーマンスといえば、もう決まっている。円周率πの計算だ。

πが計算できるものだとは思っていなかった人もいるだろうが、πを計算する方法は、じつにたくさんある。

なかでももっとも古典的な方法が、右ページの上、円に内接する6角形からはじめる方法だ。これは、あの紀元前3世紀ころの哲学者アルキメデスが使った方法と部分的におなじだ。

アルキメデスは、直径1の円(円周がちょうどπになる)に内接する正96角形の周の長さと、外接する正96角形の周の長さとのあいだにπの値があることを発見し、3.14まで正確に求めた。

右のプログラムは、そのうちの内接する多角形(正6×2^n角形、96角形はそのうちの1つ)の周を次々に求めていくものだ。プログラム中に見えるSQRは平方根を求める関数で、これはピタゴラスの定理(三平方の定理)を利用しているため。あとは小学校の算数でわかる。

LOG　logarithm(ロガリズム)の頭3文字から。BASICの組みこみ関数の1つ。数学でいう、対数のこと。ただし、対数には10を底とする常用対数と、超越数e(2.718281……)を底とする自然対数の2種類があるが、MSXに組みこまれているLOG関数は、後者のほうである。
自然対数とは、超越数eをy乗するとxになる場合、つまり、
x＝e＾y
が成り立つとき、
y＝LOG(x)
という式が成り立つような関数である。

EXP　exponent(エクスポーネント)の頭3文字から。BASICの組みこみ関数の1つ。指数と訳される(ちょっと事情が複雑だが)。
BASICの組みこみ関数としてのEXPは、
x＝e＾y
のとき、
x＝EXP(y)
が成り立つような関数。つまり超越数eをy乗にした値を返す。
正直にいって超越数eがどう役に立つのかは担当者は不勉強にして知らないが、微分積分の計算に便利なのだそうだ。そう書いてありました。

こぼれ話　ファンダムの常連GENさんからπを2300桁まで計算可能な1画面タイプの投稿があった。計算結果のセーブ、ロードまで付いている優秀な作品だったが、ピクニックとバッティングしたため今月は保留。来月、誌面の余裕とリクエストがあれば掲載を検討します。　(ピクニック担当者)

円に内接する正6×2ⁿ角形によるπの計算

正6角形

正12角形

正24角形（もうほとんど作図不可能）

正48角形

正96角形

そしていつかは円になる、かもしれない。

y　x　a　$\frac{1}{2}$　$\frac{1}{2}$

$$\begin{cases} x^2+a^2=\left(\frac{1}{2}\right)^2 \\ y^2=x^2+\left(\frac{1}{2}-a\right)^2 \end{cases}$$

↓

$$y=\sqrt{\frac{1}{2}-\sqrt{\left(\frac{1}{2}\right)^2-x^2}}$$

直径1の円を考えると、その円周はちょうどπになる。だから、直径1の円の円周を計算すれば、πを求めたことになる。そこで、アルキメデスは正6角形からはじめた。円に内接する正6角形は不思議にも1辺が円の半径とおなじになる。つまり、直径1（半径$\frac{1}{2}$）の円に内接する正6角形の1辺は$\frac{1}{2}$である。したがって、周は$6\times\frac{1}{2}=3$。これがπへの第1歩の数となる。
この正6角形を足がかりにして、正12角形を考える。正12角形の1辺の長さをyと置き、正6角形の1辺の長さの2分の1をxと置く。もう1つ、計算のつごう上、図のような線分の長さをaと置く。すると、ピタゴラスの定理から、左の2つの式が出る。この2つの式でaを消去すると、その下にあるようなyとxの関係が明らかになる。これは正6角形と正12角形の関係にとどまらず、正12角形と正24角形、正24角形と正48角形の関係でもまったくおなじ数式が成り立つ。
つまり、最初のxの値がわかればあとはイモヅル式に計算できるのだ。しかも、最初のxの値は$\frac{1}{2}$の2分の1、つまり$\frac{1}{4}$（0.25）だとわかっている。そこからどんどん計算して下の数字がずらっと出てくるのだ。

最初、このプログラムでぴたりとπが計算できるかに思われたが、プリンタに打ち出した結果を見ればわかるように、それまで順調に正しいπの値に近づいていっていたものが、正2万4576角形あたりから狂いだし、じつは、正314万5728角形以降もそのまま計算させていると、最終的に0になってしまうのだ。Xがどんどん小さくなっていって、精度の問題で起こる誤差がどんどんたまって致命的になっていくせいだろう。

それでもとちゅうでやめれば、小数点以下6桁まで、このかんたんなプログラムで求められる。計算に必要な時間は数秒である。アルキメデスが最終的に得たπの値は3.14163だったそうだから、2千年以上もの時間のへだたりは大きい。

リスト1：正6×2ⁿ角形方式のプログラム

```
10 COLOR 15,0:SCREEN 0:WIDTH 40
20 X=.25:N=6
30 PRINT N;"ｶｯｹｲ",2*N*X
40 Y=SQR(.5-SQR(.25-X^2)):N=N*2
50 X=Y/2:GOTO30
```

実行結果をプリンタに打ち出すと

```
6 ｶｯｹｲ          3
12 ｶｯｹｲ         3.10582854123
24 ｶｯｹｲ         3.1326286132819
48 ｶｯｹｲ         3.1393502030486
96 ｶｯｹｲ         3.1410319508931
192 ｶｯｹｲ        3.1414524723285
384 ｶｯｹｲ        3.1415576080383
768 ｶｯｹｲ        3.1415838922002
1536 ｶｯｹｲ       3.1415904631622
3072 ｶｯｹｲ       3.1415921061353
6144 ｶｯｹｲ       3.1415924816287
12288 ｶｯｹｲ      3.141592481629
24576 ｶｯｹｲ      3.141592481629
49152 ｶｯｹｲ      3.1415874830532
98304 ｶｯｹｲ      3.1415780626401
196608 ｶｯｹｲ     3.1415749865804
393216 ｶｯｹｲ     3.1414119511148
786432 ｶｯｹｲ     3.1408089539884
1572864 ｶｯｹｲ    3.1356849603142
3145728 ｶｯｹｲ    3.145728
```

ライプニッツの公式　同時にグレゴリーという人も発見したので、グレゴリー・ライプニッツの公式ともいう。
マチンの公式　筆算で計算するときにも実用的で、じっさいに筆算だけで500桁以上計算した人もいる。
リスト4で使用するおもな変数
N＝計算・表示する総桁数
M＝退屈しのぎのための表示用区切り
M(Ø～1, n)＝マチンの公式の2つの無限級数(分母が5の倍数のものをSØ、239の倍数のものをS1とする)を計算するにあたって、M桁ずつの精度を求めるために必要な計算項数
X、Y、Z＝上の配列Mを求めるための計算用(ここでLOGなどを使っているが詳しい説明は省略)
H＝倍精度型の配列1つにつき9桁の数を管理させるときの必要な配列数。
U＝配列1つ1つの内容を9桁で切るための定数
S(Ø～1, n)＝無限級数SØ、S1を計算するためのそれぞれの配列。配列宣言のときに1つ多めにしてあるのは、計算のとちゅうで添字が1つ余分に使われるため
T(Ø～1, n)＝SØ、S1を計算するための配列。それぞれ、分母が5、239のべき数となっている数で、次の項に行くたびにD(Ø～1)で割り算される
P(n)＝πの値を入れる配列
D(Ø～1)＝T(Ø, n)、T(1, n)を計算するための割り数
D＝割り算サブでの割り数の一時変数
K＝計算している無限級数が第何項か
F＝計算した項のプラス、マイナス
C＝桁数(表示用)
G＝配列番号(表示管理用)
Q＝商
R＝あまり

πにかぎりなく近づいていく数式

■πの値を生み出す公式

πを求める公式として知られている代表的な2つの公式がある。1つは、ライプニッツの公式、もう1つがマチンの公式だ。とくに、ライプニッツの公式は、見た目にかんたんそうで、覚えやすいためか、名前は知らなくてもなにかで見たことがあるという人は多い。それに比べてマチンの公式のほうは、一見して複雑で、ちょっと目を離したすきに忘れてしまう。

だからといって、ライプニッツの公式で、πを求めようとするとたいへんだ(リスト2)。

いや、プログラムはかんたんにできる。この2つの公式のような形を、無限級数(マチンの公式のほうは2つの無限級数が使われている)といい、たとえば、ライプニッツの公式のほうは、1を第1項、$-\frac{1}{3}$を第2項……とすると、第n項は、BASICの書式で書くと

1／(2＊n－1)＊(－1)^(n＋1)

となる。nに1とか2を代入してみればうまくいっていることがわかるだろう。nを1からはじめて1ずつ増やし、次々に加えていけばいい。それは、じょじょに4分のπに近づくだろう。したがって、各項をあらかじめ4倍していればπに近づく。ただし、うんとゆっくり。

■リスト2の動き方

リスト2を走らせると、上にBASICの組みこみ関数の1つATN(1)から計算したπの値が表示され、その下にライプニッツの公式に従って次々に無限級数を計算した現時点での値が表示される。ひじょうに速くその値は揺れ動いているため、はじめ小数第2桁以降は数字が読めない。そのうち、第2桁が確定して「3．14……」と読めるようになり、長い時間をかけて1桁ずつ、見える数字が増えていく。しかし、リスト2の最初のバージョンでは、プラスマイナスのくりかえしが(－1)^(n＋1)だったのでnが32767に達すると、エラーが出てしまった。そこで、プラスマイナス1を変数Sでくりかえすことにしてもう一度やってみた。プログラムを走らせて、数人で食事に出かけ、自衛隊の海外派遣と政治家のレトリックの貧困さについて話しながらお茶を飲み、2時間後に編集部にもどってきたときはすっかり忘れていた。しばらくべつの仕事をして、そしてふと気づいてMSXの画面を見たら……まだ、「3．14159」までが確定したところだった。しかも、そのMSXとは、あのturboRなのだ。いや、タボちゃんが悪いのではない。悪いのはこの公式だ。「πの実際の計算には役立たない」というのが、この公式に対するむかしからの世間の評価だ。実用的にはアルキメデス方式のほうがよっぽど役に立つ。

■マチンの公式で感動しよう

ところが、マチンの公式のほうは感動的なほど速くπの値に近づく(リスト3)。ふつうのMSXでも、あっというまに倍精度型の範囲内(14桁表示)で正確なπの値を計算してくれるのだ(ストップするのは239のべき乗を計算しているうちにOverflow(オーバーフロー)エラーを起こしてしまうからだ)。これを見たときの感動が今回のこの記事に結びつい

πを求める有名な公式

●ライプニッツの公式

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\frac{1}{13}\cdots\cdots$$

●マチンの公式

$$\frac{\pi}{4}=4\cdot\left(\frac{1}{5}-\frac{1}{3\cdot5^3}+\frac{1}{5\cdot5^5}-\frac{1}{7\cdot5^7}\cdots\cdots\right)-\left(\frac{1}{239}-\frac{1}{3\cdot239^3}+\frac{1}{5\cdot239^5}-\frac{1}{7\cdot239^7}\cdots\cdots\right)$$

リスト2：使えないライプニッツの公式

```
10 COLOR 15,0:SCREEN 0:WIDTH 40:KEYOFF
20 C=1:P=0:S=1
30 LOCATE 0,5:PRINT USING"ATN(1)*4 = #.#
#########";ATN(1)*4
40 P=P+4/(2*C-1)*S
50 LOCATE 0,9:PRINT USING"NOW      = #.#
#########";P
60 C=C+1:S=-S
70 GOTO 40
```

リスト3：天才的なマチンの公式

```
10 COLOR 15,0:SCREEN 0:WIDTH 40:KEYOFF
20 C=1:P=0
30 LOCATE 0,5:PRINT USING"ATN(1)*4 = #.#
#############";ATN(1)*4:LOCATE 0,9
40 CC=2*C-1
50 S1=1/(CC*5^CC)
60 S2=1/(CC*239^CC)
70 P=P+4*(4*S1-S2)*(-1)^(C+1)
80 PRINT USING"NOW       = #.#############
#";P
90 C=C+1:GOTO 40
```

↑揺れ動き続けるライプニッツの公式

↑あっというまに値が確定するマチンの公式

ているといっていい。こんなちょっとした計算をくりかえすだけで、またたくまにπを計算してしまうなんて。やっぱり、コンピュータはやり方(つまりプログラムやアルゴリズム)しだいなのだ。

リスト3があまりに速く結果を出したので、もっと大きな桁数、たとえば小数点以下500桁まで計算してみたのがリスト4だ。500桁ぜんぶを計算するまで待っているのはさすがに退屈なので、5桁ぶん確定するたびにそれまで確定したπの値すべてを表示しなおしている(見た目には5桁ずつ増えるように見える)。

500桁の数値(πの整数部分を除く)は、配列S(n, m)に9桁ずつ区切って入れて計算している。この手法は、MSXをコンピュータらしく使うためにはたいへん重要なテクニックなので、リクエストと機会があれば、これをテーマにやってみたい。

じっさいに時間をはかってみたところ、ナショナルFS-5000で約1時間6分、A1ST(turboR)で約11分かかった。いや、11分しかかからなかったというべきか。

なにしろ、世界ではじめての電子計算機といわれるENIAC(エニアック)が1949年にπの値を2037桁まで計算したときは70時間もかかったのだそうだから。リスト4の行30にある「N=500」(総桁数)を「N=2037」に変えて「M=5」も「M=2037」にして走らせ答えを出すまで見守れば、精神修養に役立ちそうだ。A1STで試したら2時間28分かかったので、たぶんふつうのMSXなら15時間近くかかるのだろう。

⬆約1時間かけて計算したπ500桁

リスト4：πを500桁まで計算するぞ!

```
10 COLOR 15,0:SCREEN 0:WIDTH 40:KEYOFF
20 DEFINT A-C,E-N
30 N=500:M=5:NM=(N-1)¥M+1:DIM M(1,NM)
40 X=2/LOG(10):Y=LOG(5)*X:Z=LOG(239)*X
50 FOR I=1 TO NM:M(0,I)=M*I/Y+1.5:M(1,I)
=M*I/Z+1.5:NEXT
60 H=N¥9+1:U=10^9:DIM S(1,H+1),T(1,H),P(
H),D(1):T(0,0)=4*4*5:D(0)=5^2:T(1,0)=4*2
39:D(1)=239^2
70 FOR I=1 TO NM
80 FOR J=0 TO 1
90 FOR K=M(J,I-1)+1 TO M(J,I)
100 FOR A=0 TO H:S(J,A)=T(J,A):NEXT
110 D=D(J):GOSUB 1000
120 FOR A=0 TO H:T(J,A)=S(J,A):NEXT
130 D=2*K-1:GOSUB 1000
140 IF (K+J)MOD2 THEN F=1 ELSE F=-1
150 GOSUB 2000
160 NEXT K:NEXT J
170 C=0:G=0:LOCATE 0,0:PRINT "π =";STR$(
P(0));".";
180 G=G+1:A=0:A$=RIGHT$(STR$(P(G)+U),9)
190 A=A+1:C=C+1:PRINT MID$(A$,A,1);
200 IF C MOD 5  =0 THEN PRINT " ";
210 IF C MOD 25 =0 THEN PRINT:PRINT SPC(
6);
220 IF C=N THEN END
230 IF C=M*I THEN NEXT I
240 IF A=9 THEN 180 ELSE 190
1000 '●●● WARIZAN SUB
1010 FOR A=0 TO H
1020 Q=INT(S(J,A)/D)
1030 R=S(J,A)-Q*D
1040 S(J,A)=Q
1050 S(J,A+1)=S(J,A+1)+R*U
1060 NEXT:RETURN
2000 '●●● KAGEN SUB
2010 FOR A=H TO 0 STEP -1
2020 P(A)=P(A)+(S(J,A)+E)*F:E=0
2030 IF P(A)>=U OR P(A)<0 THEN E=1:P(A)=
P(A)-U*F
2040 NEXT:RETURN
```

COPY文を使ったアニメーション技法

BASICテクニック

#19 手作りの紙芝居のために

MSX2以降なら裏のページとCOPY文を使ってかんたんにアニメーションのプログラムを作ることができる。その仕組みとサンプルを紹介しよう。

ページ　SCREEN5のVRAMでページといえば、32Kバイト単位のVRAMの領域を指す。SCREEN5のVRAMには、ページ0～ページ3の4ページがあり、ふつうは、ページ0に絵をかいたり、その絵を画面に表示したりしている。

SETPAGE文　SCREEN5以上で使えるステートメント。
n、mを0～3の整数とすると
SETPAGE　n，m
は、ページnを画面に表示し、グラフィック命令などによるかきこみはページmにおこなう、という意味になる。

COPY文　これにはいろんな使い方があってぜんぶ説明するとややこしい。今回の記事で使った機能だけを説明すると、このステートメントは、VRAMのある部分からある部分にデータを複写することができる。しかも、そのときに、ページを指定してページ間を超えて複写することができるのだ。
この機能に関する書式を示すと、
COPY(x1,y1)−(x2,y2),n　TO　(x3,y3),m
この文は「ページnの座標(x1,y1)から座標(x2,y2)までのデータを、ページmの座標(x3,y3)からはじまる部分に複写する」という意味になる。なお、ページがちがっても座標系の数値は共通。

SCREEN5の3枚の裏ページ

今回の記事とプログラムは、VRAM128KのMSX2とそれ以降の機種が対象になる。なにしろ、SCREEN5をページ0から3まで使うし、COPY文が主役になるからだ。

MSX1の人はごめんなさい。VRAM64Kの人は、9枚アニメならこの記事と同様の原理でアニメーションできる。

といっても、今ではほとんどの人がこの条件を満たすはず(Mファンのアンケート調べでは95パーセント)。

■SCREEN5のページ

VRAM128Kの場合、SCREEN5には4枚のページがある。ページとは、メインRAMとかディスクとかではまたちがった使い方をするが、ここでは画面1枚に対応するVRAMの領域のことだと思っていてまちがいない。

たとえば、ごくふつうのSCREEN5のプログラムの場合、ページ0のVRAMの内容が画面に表示されている。LINE文などでグラフィックをかくと、ページ0のVRAMの内容が変化し、それが画面の変化となって現れる。

ところで、SCREEN5のページ0のVRAM容量は、アドレスにして0～&H7FFF、32Kバイトだ。VRAMは128Kあるというのにあと96KバイトぶんのVRAMはどうしているのだと思うだろう。SCREEN7とか8ではもっと事情がちがうがここではSCREEN5にかぎって話を進める。

画面にして3枚ぶんのVRAMは、ふだん表の画面の裏でただ眠っているのだ。

だから、SETPAGE文を使って、ページ1やページ2を画面に表示させてみるとわけのわからない模様が入っている。これは使われていないからCLSさえもされていないということなのだ。気になる人は、
SCREEN5：FOR I＝1 TO3：SETPAGE，I：CLS：NEXT
を実行すれば、きれいになる。きれいになったからといって、とくにいいことはないが。

■図1 アニメーションの仕組み

SCREEN5は、横256ドット、縦212ドットで、右端と下端に多少あまりができるが、X座標は85、170、Y座標は70、140のところでキリよく分割した。この9分の1画面の絵をページ0の中央に次々に表示するとパラパラアニメの原理で動いてみえるわけだ。ページ1～3にこの図のような連続した絵をかきこみ、数字の順に次々とページ0の中央にCOPYしてくれば、30～33ページの下をずっと流れている「3原色の円によるデモ」になるわけだ。では、そうするためにはいかにプログラムを組めばいいか。32、33ページのリスト1、2がその解答の1つだ。

※なお、この図のページ1～3の写真には各区画を区別する白い線が入っているが、リスト1、2ではそういう線をかいてはいない。

ページ1　85　170　70　140
1 2 3 4 5 6 7 8 9

ページ2　85　170　70　140
10 11 12 13 14 15 16 17 18

ページ3　85　170　70　140
19 20 21 22 23 24 25 26 27

ページ0（テレビ画面）
(85, 70)

■3枚のページを控室に使う

いずれにしてもこのふだん使われていないページ1～3という裏ページを使って、アニメーションができるのだ。

アニメーションとは、ようするに高速紙芝居、つまりパラパラアニメだ。ここでは、1枚のページを9つの区画に分け、その1区画に1コマぶんの絵をかくことにした。そうすると、9×3で27コマの絵がかける。

この27コマの絵をページ0(ふだんから見えている画面)の中央に、次々に切り換えてコピーしてくれば、パラパラアニメの要領で動いて見えるし、物語のあるコママンガをかいておけば、自動紙芝居になる。

以上の構造を図解すると上のようになり、実際にそういうプログラムを動かすとこの30ページから33ページまで下を流れている写真のようになるわけだ。

リスト1：個々の絵の作成

```
10 COLOR 15,0,0:SCREEN 5,0:DEFINT A-Z
20 A=85:B=70:R=20
30 F$="SAMPLE"
40 DIM X(8),Y(8),XX(2),YY(2)
50 FOR I=1 TO 7:COLOR=(I,SGN(I AND 4)*7,
SGN(I AND 2)*7,SGN(I AND 1)*7):NEXT
60 FOR I=0 TO 2:FOR J=0 TO 2:X(I*3+J)=A*
J:Y(I*3+J)=B*I:NEXT:NEXT
70 FOR I=0 TO 2:CIRCLE (R+I*50,R),R,2^I:
PAINT STEP(0,0),2^I:NEXT
80 FOR I=0 TO 26
90 FOR J=0 TO 2:READ XX(J),YY(J):COPY (J
*50,0)-STEP(40,40) TO (XX(J),YY(J)),,TOR
:NEXT
100 COPY (A,B)-STEP(A,B),0 TO (X(I MOD 9
),Y(I MOD 9)),I¥9+1
110 LINE (A,B)-STEP(A,B),0,BF
120 NEXT
130 FOR I=1 TO 3:SETPAGE I,I:BSAVE F$+MI
D$(STR$(I),2),0,&H7FFF,S:NEXT
140 DATA  97, 56, 87,108,137, 90
150 DATA  93, 59, 92,110,136, 85
160 DATA  90, 63, 97,110,134, 80
170 DATA  88, 68,102,110,131, 76
180 DATA  87, 73,107,108,127, 73
190 DATA  88, 78,111,104,122, 71
200 DATA  90, 83,114,100,117, 70
210 DATA  93, 87,116, 95,112, 71
220 DATA  97, 90,117, 90,107, 73
230 DATA 102, 92,116, 85,103, 76
240 DATA 107, 93,114, 80,100, 80
250 DATA 112, 92,111, 76, 98, 85
260 DATA 117, 90,107, 73, 97, 90
270 DATA 115, 88,107, 75, 99, 88
280 DATA 113, 87,107, 77,101, 87
290 DATA 111, 86,107, 79,103, 86
300 DATA 109, 85,107, 81,105, 85
310 DATA 107, 84,107, 84,107, 84
320 DATA 106, 81,105, 86,110, 84
330 DATA 105, 78,103, 88,113, 85
340 DATA 104, 75,101, 91,116, 85
350 DATA 103, 72, 99, 93,119, 86
360 DATA 102, 70, 97, 96,122, 87
370 DATA 101, 67, 95, 98,125, 87
380 DATA 100, 64, 93,100,128, 88
390 DATA  99, 61, 91,103,131, 88
400 DATA  98, 58, 89,105,134, 89
```

■リスト1の解説

・おもな変数

A、B＝アニメ用の絵1コマの区切り用。それぞれ、85、70の数値が入る。31ページ図1参照

R＝デモに使う円の半径

F＄＝画面セーブ用のファイル名。実際には「SAMPLE1」のように末尾に数字を加えて使う

X(n)、Y(n)＝nは0～8の各ページにおけるアニメ用コマの表示順に応じた座標。たとえば、座標(X(4),Y(4))は、あるページで5番目に使われる区画の座標

XX(n)、YY(n)＝nは0～2で、それぞれ赤、青、緑の3原色の円に対応する。アニメ用のコマ絵をかくときの各コマごとの円の複写先座標

・プログラム解説

10～40　初期設定

50　パレットコード1～7をパレット設定。たとえば、パレットコード4の色を純粋な赤(R、G、BのRだけを7に設定)にする。この処理によって、赤と緑の円をOR演算で重ねると、重なった部分の色が光の3原色の原理に従って黄色になる、などの関係を再現した

60　X(n)、Y(n)の設定

70　画面の上部に50ドットずつ離して3つの色の円をかき、内部を色で塗る。パレット設定のために左から青、緑、赤の円が現れる

80　27枚ぶんのコマ絵の作成開始

90　行140からの座標データを読み出して3原色の円を画面の中央に複写。そのさい、透明色は無視し、かつ色が重なった部分はOR演算の処理をする(ロジカルオペレーションのTOR)

100　画面中央(これはページ0)にできた絵をページ1～3に順に複写する

110　画面中央の1区画ぶんを透明色で塗りつぶす(消しゴムをかける)

120　行80と対応するＮＥＸＴ。行80から行120までを実行しおわるとページ1～3に31ページの図1のような絵ができあがる

130　ページ1～3の絵を「SAMPLE1」～「SAMPLE3」というファイル名でディスクにセーブ

140～400　3原色の円のアニメーション用座標データ。このデータにしたがって円を表示すれば連続した軌跡をたどるようになっている

■リスト2の解説

・おもな変数

リスト1とおなじ

・プログラム解説

1000～1030　リスト1と同様

1040　「SAMPLE1」～「SAMPLE3」をページ1～3にロード

1050　アニメを見せるための枠作成。周囲を深緑で塗る

1060　27枚のアニメーション開始

1070　31ページ図1のようにページ1～3の各部分から順に取り出し、ページ0の画面中央に複写

1080　時間待ち。Wの終値を変えればアニメーションのスピードが変わる

1090　行1060と対応するNEXT

1100　行1060からくりかえし

■3原色の円のデモ

リスト1はアニメ用の各コマの絵をかき、それをディスクにセーブするところまで。リスト2は、セーブされた画面をページ1〜3に読み出して、アニメーションをおこなう。

光の3原色(赤、緑、青)のスポットが重なったり離れたりするとこんなふうに色が変わるはずだ。使用するパレットコードを0〜7に限定し、この範囲で光の3原色の原理がロジカルオペレーションのOR演算に合致するようにパレット設定する必要があり、プログラムはリスト1の行40だが、ここを組むまえに頭のなかに描いた設計図を参考までに示すと図2のようなものだった。

■ディスクドライブがない場合

ところで、リスト1、2をこのままで使うにはディスクドライブが必要だが、ディスクドライブがなくても適切な行をけずってリスト1とリスト2を結合すればちゃんと動く。ディスクドライブがあっても、その結合したものを使うほうがいい。

結合する部分は、〈行30と行130を除くリスト1〉+〈リスト2の行1050〜1100〉。これでディスクを使うことなく、3原色のアニメデモが見られる。

リスト2:27枚の絵でパラパラアニメ

```
1000 COLOR 15,0,0:SCREEN 5,0:DEFINT A-Z
1010 A=85:B=70:F$="SAMPLE"
1020 FOR I=1 TO 7:COLOR=(I,SGN(I AND 4)*
7,SGN(I AND 2)*7,SGN(I AND 1)*7):NEXT
1030 FOR I=0 TO 2:FOR J=0 TO 2:X(I*3+J)=
A*J:Y(I*3+J)=B*I:NEXT:NEXT
1040 FOR I=1 TO 3:SETPAGE I,I:BLOAD F$+M
ID$(STR$(I),2),S:NEXT:SETPAGE 0,0
1050 LINE (A-1,B-1)-STEP(A+1,B+1),14,B:P
AINT (0,0),12,14
1060 FOR I=0 TO 26
1070 COPY (X(I MOD 9),Y(I MOD 9))-STEP(A
-1,B-1),I¥9+1 TO (A,B),0
1080 FOR W=0 TO 200:NEXT
1090 NEXT
1100 GOTO 1060
```

■リスト2だけの利用法

リスト1と2はもともと1本だったが、ほかの方法でかいた絵を使って同様のアニメができるようにアニメ部分だけをリスト2として独立させたのだ。

リスト2は、「SAMPLE1」〜「SAMPLE3」というファイル(SCREEN5の画面データ)を、それぞれページ1〜3に読みこみ、それを使ったアニメを見せる。

31ページの図1のような9コマずつに分かれたCGをかいて、それを適切なファイル名でセーブしたディスクを作っておけば、別の絵を使ったアニメができるわけだ。行1020で3原色デモ用のパレット設定をしているのでこの行は消去しておく。

行1010のF$の中身の設定と行1040を変更すればさまざまなファイル名に対応できるし、NAME"〈現在のファイル名〉"AS"〈希望のファイル名〉"というコマンドを使って、すでにセーブされているファイル名のほうを変更してもいい。

■図2 光の3原色を見せるパレット

※カッコ内の数値がパレットコード

リスト1とリスト2の確認用データ

使い方は54ページ

●リスト1

10>ad40	20>gR20	30>lI10	40>3s30
50>xrb0	60>e3b0	70>94M0	80>a310
90>Zqh0	100>WtL0	110>pV30	120>B200
130>oOQ0	140>DK40	150>lI40	160>4G40
170>gI40	180>aL40	190>bH40	200>tG40
210>QG40	220>sG40	230>nH40	240>ND40
250>AG40	260>UE40	270>aJ40	280>1J40
290>XJ40	300>HI40	310>6J40	320>ZG40
330>1J40	340>NH40	350>IJ40	360>cH40
370>dJ40	380>iJ40	390>5I40	400>TM40

●リスト2

1000>ad40	1010>Yx40	1020>xrb0	1030>e3b0
1040>ZkS0	1050>YVD0	1060>a310	1070>EhQ0
1080>8A20	1090>B200	1100>l600	

プログラマブルな10個のキー

BASICテクニック

#20 ファンクションキーの底で光るもの

こんなに便利なものが目の前にあるのにほとんどの人が活用していない(ような気がする)ファンクションキー。その底で光るものを見つめてみたい。

ファンクションキーをめくって

ファンクションキーは、フルネームを「プログラマブル・ファンクションキー」というくらいで、基本的にプログラマブルなものだ。プログラマブルというのは、プログラマのふりをするという意味ではもちろんなく、日本語にくだくと「ユーザーの好みにあわせて内容を変更できる」という意味になる。

ファンクションキーの設定はやり方がかんたんだし、効果もたちどころに現れるので、BASICで遊びはじめた初期にはいろいろもてあそばれるものだ。

KEY1, 〈文字列〉

とすればF1キーの内容を〈文字列〉に変更できる。15字以内と

プログラマブル・ファンクション・キー　programable function key。MSXでは「F」ではじまるキーだが、ほかのパソコンでは「PF」と表示されているものもある。この場合のPが、この「programable」のPだ。

15字以内　F1～F10にはそれぞれメモリの16バイトずつが割り当てられていて、それぞれのキーの先頭のアドレスは決められたとおりで変化することはない。また、各キーの内容の16バイト目はかならず「Ø」が入るようになっている。この「Ø」は区切り用で、ファンクションの内容の途中に「Ø」があると、それ以降の内容はファンクションキーを押しても出ない(ファンクション表示には出る)。逆に、16バイト目にPOKE文でØ以外のデータを書きこんでおくと、そのキーを押すことで次のファンクションキーの内容もいっぺんに出すことができる。

F1

10個のファンクションキーに対して、メモリの&HF87F以降の160バイトが内容保存用に確保されている(1個あたり16バイトずつ)。たとえば、初期状態におけるF1キーの場合は、図のような数値(ただし数値は16進数。たとえば「2Ø」は&H2Ø、つまり32のこと)が入っている。これはキャラクタコードで、文字に変えていくと「color」と空白1個。あとは「ØØ」で埋められ(とくに16バイト目はかならず「ØØ」になる)、17バイト目からはF2キーの内容がはじまるのだ。

63	6F	6C	6F	72	2Ø	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ

&HF87F　16バイト

■F1キーはアドレス&HF87F以降16バイトにいる

いう制限付きだが、ひらがなでもカタカナでも、CHR$関数を使ったコントロールコードなどでもかまわない。

　ファンダムでよく使っているのは、カーソルキャラクタ(■)が入ったプログラムを打ちこむときだ。カーソルキャラクタはようするにカーソルになっているあいつだが、そもそも打ちこむためのキャラクタではないから、それを打ちこむキーというのはもともとない。そこで、KEY1，CHR$(255)を実行して、F1キーの内容をカーソルキャラクタにしてしまうわけだ。もっとひどいのが「スペースでない空白」という別名を持つCHR$(254)で、これは見た目がなにもなくて、それを打ちこむキーもやはりない。こういうなにからなにまでない文字でさえ、ファンクションキーに設定してリスト中に文字データとして入れることができるのである。

　ところで、この器用なファンクションキーは、どこに自分自身の中身をたくわえているのか。F1キーをめくってみると、それはメモリの&HF87F以降16バイトの領域だった(MSX1からターボRまで共通)。ここに入っている数値はF1キーの内容になっている文字1つ1つのキャラクタコードなのだ。

　ファンクションキーの中身はKEYLISTでいちおうわかることになっているが、この場合、コントロールコードはすべて空白に置き換えられて表示されるのであまり意味がない。

　そこでコントロールコードを目に見える形(色付きの文字)で表示するために作ったのがリスト1だ。色付きの文字は、CTRLキーとその文字のキーを押すことを意味している。

　初期状態でリスト1を実行した状態が上の写真だ。たとえば、F10の中身は、CTRL+L、RUN、リターンという一連のキー操作であり、このとおりにやってみれば、F10の中の機能の仕組みがよくわかるだろう。

⬆リスト1の実行後の画面。赤地に黄の文字がコントロールコードを表している。

リスト1の解説

■おもな変数

A$(0)～A$(9)＝各ファンクションキーの中身。このプログラムの目的からすれば、配列を用意する必要はないが、ファンクションキーの中身をランダムに入れ換えて遊んだりする場合は配列に入れておいたほうが便利なのであえてこうした

AD＝ファンクションキーの内容が保持されているアドレスの先頭番地

CH＝調べているアドレスに入っている数値データ

■プログラム概説

10～20　初期設定

30　グラフィックキャラクタにコントロールコードに対応するパターンを定義して色を付ける

40～100　ファンクションキーの内容を保持しているメモリ領域から1バイトずつ取り出して、各ファンクションキーごとに内容を配列A$に連結して入れる。とちゅうにデータ「0」があると、次のファンクションキーへ移る

110～170　A$(0)～A$(9)を「　」でくくって表示

ただし、コントロールコードは行140でグラフィックキャラクタに変換して表示している。たとえば色付きの「M」は、CTRL+M、つまりリターンコードのこと。DELのみ「@」で代用

リスト1：ファンクションキーの底をのぞく

```
10 COLOR 15,4,7:SCREEN 1:WIDTH 29:KEYON
20 DEFINT A-Z:AD=&HF87F:DIM A$(9)
30 FOR I=Ø TO 255:VPOKE I,VPEEK(512+I) X
OR 255:NEXT:FOR I=Ø TO 3:VPOKE 8192+I,&H
8B:NEXT
40 FOR I=Ø TO 9
50 FOR J=Ø TO 15
60 CH=PEEK(AD+I*16+J)
70 IF CH=Ø THEN J=15:GOTO 9Ø
80 A$(I)=A$(I)+CHR$(CH)
90 NEXT J
100 PRINT USING"F##="+CHR$(162);I+1;
110 FOR J=1 TO LEN(A$(I))
120 A$=MID$(A$(I),J,1)
130 IF A$<" " THEN A$=CHR$(1)+CHR$(ASC(A
$)+&H4Ø) ELSE IF A$=CHR$(127) THEN A$=CH
R$(1)+CHR$(&H4Ø)
140 PRINT A$;
150 NEXT J:PRINT CHR$(163)
160 NEXT I
```

プログラム確認用データ 使い方は45ページ

```
 10>pW4Ø    20>Gb4Ø    30>sBaØ    40>tØ1Ø
 50>n11Ø    60>gA4Ø    70>Qe3Ø    80>XX4Ø
 90>v6ØØ   100>s56Ø   110>ØI3Ø   120>jE2Ø
130>bu31   140>3EØØ   150>Q22Ø   160>s6ØØ
```

不思議で便利で貪欲なINPUT$

たとえば、「SCREEN0：WIDTH40」といった、よく打ちこむ文字を設定しているだけでもファンクションキーは有用だが、コントロールコードの知識があれば、ファンクションキーは飛躍的に便利になる。

コントロールコードについては、1990年の6月号でやったので、ここでは欄外にかんたんな一覧を載せるだけにしておく。

■CHR$関数による設定

ファンクションキーにコントロールコードを設定するにはどうすればいいのか。

おもしろみはないが、安全確実なのが、CHR$関数と、その拡張版のSTRING$関数による設定だ。

たとえば、33ページのリスト2がそうだ。これは、じっさいにBASICピクニック担当者が使っているファンクションキーの設定用プログラムだ。

いずれにしろ、このやり方は、多少プログラムの組める人にとってはとくに目新しいところがないし、ファンクションキーの設定をプログラムの形で残すとすれば、とりあえずはこういうやり方以外にないだろう。

■意外なやり方

しかし、もう1つ、意外なやり方がある。それを知ったのは、まだ、わたしがBASICを覚えはじめたころで、手探りでちょっとしたプログラムを作っているときだった。それは、たくさんのDATA文がならぶプログラムで、あれやこれやいじっているうちに、数行おきにDATA文を削除する必要が生じた。数行ぶんまとめて消すならDELETE文があるし、1行だけなら行番号だけを打ちこんでリターンキーを押せばいい。しかし、リストを見ながら、不要な行を見つけ、べつの場所に行番号を打ちこみ、リターンキーを押し、その行を抹消していくという作業はかなりめんどうなものだった。5行くらい消したあと、わたしはその作業にあきて近くにいた当時の副編NOP(ノップ)に聞いたものだ。

「1行をかんたんに消す方法はないの?」

すると、まずNOPはCTRL＋Eを教えてくれた。これである文字以降をいちどに消せるので、行番号のあとにカーソルーを持ってきて、リターンキーを押すだけでその行が消せる。

わたしは喜んで作業を続けたが、たちまちあきてしまった。手順が3つに分かれているのがめんどうでしかたがない。すると、NOPはいきなり、指でなにかを数えると、

KEY1，INPUT$(3)

と打ちこんでリターンキーを押したのだった。

「Ok」が表示されない。NOPは、CTRL＋Fを押し、CTRL＋Eを押し、そして最後にリターンキーを押した。ようやく「Ok」が出てきた。

「はい、これでF1キーでその行が消せます」

NOPはときどき信じられないような見え透いたウソをつくが、このときはほんとうだった。

■貪欲なINPUT$

わたしは文章書きという職業柄か、BASICの文字関数が好きだが、なかでもこのときNOPが教えてくれたINPUT$という関数が大好きだ。

たとえば、

A$＝INPUT$(5)

を実行すると、それ以降にキー入力された5文字をA$に入れてくれる。この関数は、カッコのなかの数だけ、入力されたキーの情報をたくわえる働きがあり、たくわえたものをA$に移したり、ファンクションキーに送ったりするわけだ。

とくに、注目すべきは、この関数が文字として受け付けてくれるキー操作は、CTRL＋STOP、STOPキーを除くすべてだという点だ。たとえば、ダブルクォーテーションやコンマはもちろん、CLS、INS、DEL、BS、そのほかCTRLと組み合わせて入力されるコントロールコードなどもほかの文字と同様に受け付けてくれる。この貪欲さがすばらしい。

■ファンクションキーの保存

この方法ならあれこれ数値を考えなくても、手を動かしながらいろいろと設定できる。気に入らなければやり直すのもかん

■便利でおもしろい手動方式

KEY1，INPUT$(3) [↵]

↓

[CTRL]＋[F]

CTRL＋Fは、次の単語の先頭にカーソルを飛ばす。行番号の頭にカーソルを置いてあれば行の内容の先頭文字にカーソルが飛ぶ。

↓

[CTRL]＋[E]

CTRL＋Eはカーソル以降、行末までの表示を消す。ここで、その行はプログラムが消え、とりあえずは行番号だけになる。

↓

[↵]

最後にリターンキーを押して、その行の息の根を完全に止める。ここまでの操作をF1キー1押しにこめているのだ。

コントロールコード　キャラクタコード0～31と127をとくにコントロールコードという。これらは通常の意味での文字ではなく、「表示」のかわりに「制御」する文字で、画面消去やビープ音の発生など、さまざまな個性的機能を持つ。BS、TAB、HOME、CLS、RETURN、INS、ESC、DEL、カーソルキーは、それぞれコントロールコード入力用の特殊キーだが、CTRLキーとフルキーのどれか1つを同時に押すことでもDELコード以外のコントロールコードを入力できる。

ここでは、CTRLキーと組み合わせて押すキー(下の一覧では【 】でくくった文字)とコントロールコード(カッコ内の数値)、およびその機能をかんたんにまとめた。この組み合わせの片方のキーは、31ページのリスト1を実行したときに表示される色付きの文字と対応している。

■おもなコントロールコード一覧

【A】(1)＝グラフィックヘッダ
【B】(2)＝カーソルを直前の語の先頭へ移動
【C】(3)＝入力待ち状態の終了
【E】(5)＝カーソル以下、行の終わりまで削除
【F】(6)＝カーソルを次の語の先頭へ移動
【G】(7)＝ビープ音を鳴らす
【H】(8)＝BSキーとおなじ
【I】(9)＝TABキーとおなじ
【J】(10)＝行送り(カーソルキーの下に似ているが、画面の最下行でこのコードを入力すると画面全体がスクロールする)
【K】(11)＝HOMEキーとおなじ
【L】(12)＝CLSキーとおなじ
【M】(13)＝RETURNキーとおなじ
【N】(14)＝カーソルを行末へ移動
【R】(18)＝INSキーとおなじ
【U】(21)＝カーソルのある行の先頭から行末までを削除し、カーソルをもとあった行の先頭へ移動
【〔】(27)＝ESCキーとおなじ
【¥】(28)＝カーソルキー右とおなじ
【〕】(29)＝カーソルキー左とおなじ
【^】(30)＝カーソルキー上とおなじ
【_】(31)＝カーソルキー下とおなじ
DEL(127)＝カーソルの指す文字を削除し、以降の文字を前に詰める。DELキー以外に入力できない

INPUT$関数　この関数は正しくは、
INPUT$(n，#m)
という書式で、nに文字数、mにファイル番号を入れて使うもので、ファイルからの入力を受け付ける関数なのだ。ただし、「，#m」を省略すると、キーボードからの入力に限定される。今回の記事では、その省略した使い方だけを扱っている。

リスト2：ファンクションキー設定のサンプル

```
10 COLOR 15,0,0:SCREEN 0:WIDTH 40
20 KEY1,STRING$(13,127)
30 KEY2,CHR$(18)+"LOAD"+CHR$(34)
40 KEY3,CHR$(21)+"FILES"
50 KEY4,CHR$(12)+"LIST"
60 KEY5,CHR$(21)+"RUN"+CHR$(13)
70 KEY6,CHR$(21)+"COLOR15,4,7"+CHR$(13)
80 KEY7,CHR$(21)+"COLOR15,1,1"+CHR$(13)
90 KEY8,CHR$(21)+"SCREEN0"+CHR$(13)
100 KEY9,CHR$(18)+"SAVE"+CHR$(34)
110 KEY10,CHR$(18)+"RUN"+CHR$(34)
```

プログラム確認用データ 使い方は45ページ

```
10>kM20   20>T410   30>2n60   40>QD30
50>9u20   60>DH60   70>tFA0   80>jCA0
90>IJ80   100>lq60  110>xt60
```

```
     LOAD"   FILES   LIST   RUN
```

↑リスト２を実行したときのファンクション表示（Ｆ１〜Ｆ５）。Ｆ１のただなにもない部分はデリートコードが13個入っている

```
COLOR1   COLOR1   SCREEN   SAVE"   RUN"
```

↑おなじくＦ６〜Ｆ10のファンクション表示。Ｆ９には、SAVEコマンドが入っているので操作に注意しなくてはいけない

たんだ。

そうやってＦ１からＦ10まで自分の好みに設定したら、やはりディスクにセーブしておきたい。ファンクションキーの内容を保持しているメモリの領域を、マシン語のセーブとおなじ要領で「BSAVE〜」すればいいのだ（下図参照）。ファイル名はてきとうでいいが、&HF87F（開始番地）と&HF91E（終了番地）だけはまちがえないように。

ロードのときは「BLOAD」。セーブのときとちがって、アドレスがいらないので楽だ。

この手法を使えば、いろんなファンクションキーセットをちがったファイル名でセーブしておき、あとで必要に応じて使いわけることができる。

リスト２の解説

■プログラム概説

10　画面設定
20　Ｆ１にDELコードを13個入れる。これは、ファイル表示させたときにファイル名と次のファイル名のあいだが13文字だから。画面の左端でＦ１キーを押すと次のファイル名が画面の左端にきてなにかと便利
30　Ｆ２に①INSキー、②LOAD、③「"」を入れる。表示されたファイル名の先頭にカーソルをあわせてＦ２キーを押すと、そのファイルをロードする準備ができる仕組み。「"」は直接文字データとしてはBASICの行のなかに存在できないので、このようにCHR$(34)を使う
40　Ｆ３に①CTRL＋U、②FILESを入れる。CTRL＋Uは、まわりにごちゃごちゃと字が表示されていても、必要最低限の文字消去をしてくれるので便利
50　Ｆ４に①CLSキー、②LIST
60　Ｆ５に①CTRL＋U、②RUN、③リターン
70〜90　Ｆ６〜８にカラー設定やSCREEN設定を入れる
100　Ｆ９に①INSキー、②SAVE、③「"」を入れる
110　Ｆ10に①INSキー、②RUN、③「"」を入れる

最後に、リセットしなくてもファンクションキーの状態を初期状態にもどす方法を教えておこう。
DEFUSR=&H3E
としておいて、
A=USR(0)
でOK。このあとSHIFTキーを押すとか、KEYONを実行するなどで、電源を入れたときとおなじファンクションキー表示が出てくる。

■現在設定されているファンクションキーの内容を保存する

```
BSAVE"FUNCTION.01",&HF87F,&HF91E
```

■そのデータを読み出す

```
BLOAD"FUNCTION.01"
```

■ファンクションキーを初期状態にもどす

```
DEFUSR=&H3E:A=USR(0)
```

マシン語への架け橋USR関数

BASICテクニック

#21 USR関数でまずなにができるか

まだ寒いがとっくに春は立った。春が来てはじめてほんとうの新しい年がはじまり、新しい心が芽生えてくる。そろそろ、マシン語をかすめてみようか。

関数がマシン語を実行する

マシン語はたしかにわかりにくい。

BASICのプログラムは、あるていど日常生活に根ざしたわかりやすさを持っている。PRINTとか、COPYとかいうステートメントには、プリントしまっせ、とか、コピーしちゃうよ、などという肉声が多少なりとも感じられ、そのステートメントの使い方を知らなくても、なんとなく安心できる。

しかし、マシン語は、命令もその命令があつかうアドレスも計算に使う数値もルリもハリもミソもクソもみんな数字でできている。じっさい、たとえば&H21という命令と&H21という数値をいったいこやつらはどうやって区別しているのだろうと怪しんだものだ。

■MSXにおけるマシン語

基本的にわかりにくい存在であるマシン語は、「ホームパーソナルコンピュータ」であろうとするMSXでは日陰者あつかいされてきた。たとえば、PC88などの他機種ではふつう内蔵されているモニタ(傍注参照)がMSXでは内蔵されていない。そのために、他機種で見られる「ダンプリスト」(マシン語プログラムリスト。傍注参照)が掲載されることはまずない。

そういうMSXであるから、オールマシン語のプログラムを打ちこむという習慣はふつうのMSXユーザーにはなく、オールマシン語のプログラムを保存したり読み出したりするための命令、BSAVEとBLOADはじっさいにはその目的で使われることがあまりない。

そのかわり、MSXではBASICプログラムがマシン語プログラムを抱きかかえている。ある行のほんの一部に短いマシン語が16進数の文字列で入っていたり、DATA文のなかにマシン語が混じっていたり、リストのほとんどがDATA文でそれぜんぶがマシン語だったり、リストのほとんどがREM文で、そのREM文がじつはマシン語だったりする。

こんなふうにMSXのマシン語はBASICプログラムのなかに溶けこんでいるのだ。溶けこみ方にはいくつかあるが、なかでももっともよく使われ、かつ有用なのがUSR関数である。わたしは個人的にこの不思議な関数が大好きだ。

前置きが長かったが、ようやく本題に入れそうだ。

■マシン語を呼ぶUSR関数

USR関数は、BASIC中でマシン語プログラムを呼び出すためのものだ。

機能的にUSR関数にもっとも近いBASICの命令は、意外かもしれないがGOSUBだ。

モニタ　マシン語の話に出てくるモニタとは、ディスプレイのことではなく、マシン語データを直接メモリに打ちこんだり、読み出したり、実行したりするツールのこと。MSXでも、マシン語入門の記事や本ではモニタのリストが掲載されていることが多い。

ダンプリスト　ある範囲のメモリの状態をリストにしたもの。ダンプリストを打ちこむには、モニタが必要。

USR関数　Mファン編集部内部では、「ユーザー関数」ではなく「ユーエスアール関数」と読んでいる。DEFFN～のユーザー定義関数とまぎらわしいためだ。
USR関数は、USR0～USR9までの10個を設定できる。USR関数の定義は、
DEFUSRn=〈アドレス〉
(nは0～9)
という書式でおこなわれ、
A=USRn(〈引数〉)
などの形でUSR関数が使われたときに定義されたアドレス以降のマシン語プログラムを実行する。なお、nが0のときはnを省略できるので、USR0とUSRはおなじもの。

BIOS　Basic Input Output System。「バイオス」と読み、基本入出力システムなどと訳されるが、早い話がMSXの入出力用のマシン語サブルーチン集でROMに内蔵されている。ふだんでも内部でこのBIOSをしょっちゅう使っているはずだが目に見えない。このBIOSを使うときは、各機能に応じたアドレスから入らなければならない。そのアドレスのことをエントリ(入口)ということがある。

■マシン語プログラムのいらないUSR関数のサンプル

①USRØの定義

```
DEFUSRØ=&H41
```

USR0を定義する画面

BIOS / USRØ → &HØØ41 画面表示の禁止 / &HØØ44 画面の表示

まず「画面表示の禁止」という機能があるBIOSエントリのアドレスをUSRØに定義。これで関数USRØを使うとここが実行される。ちなみにUSRØの「Ø」は省略することが多い。

②USR1の定義

```
DEFUSR1=&H44
```

USR1を定義

BIOS / USRØ → &HØØ41 画面表示の禁止 / USR1 → &HØØ44 画面の表示

次に「画面の表示」という機能のあるエントリのアドレスをUSR1に定義。「画面表示の禁止」という毒に対する解毒剤としてこれを定義しておかないと、リセットするしかなくなる。

③USRØを使う(BIOS呼び出し)

```
A=USRØ(Ø)
```

いよいよ決行。真っ暗

BIOS / USRØ → &HØØ41 画面表示の禁止 / USR1 → &HØØ44 画面の表示

そこで「A=USRØ(Ø)」を実行すると、定義してあった&H41が呼ばれる。この直後、ディスプレイは真っ暗になり、画面上はまったくなんの反応も起きなくなる。ほんとうは、周辺色(この場合、故意に黒にしておいた)の色だけになるので、周辺色が青の場合は真っ青になるのだが。

④USR1を使う(BIOS呼び出し)

```
A=USR1(Ø)
```

そしてふたたび見える

BIOS / USRØ → &HØØ41 画面表示の禁止 / USR1 → &HØØ44 画面の表示

そこに、解毒剤USR1。この関数を「A=USR1(Ø)」という形で使い、&H44「画面を表示」を実行。ふたたびもとの平和な世界にもどりましたとさ。めでたしめでたし。

GOSUBは、指定した行からはじまるサブルーチンを実行してもとの場所にもどってくる。USR関数も、指定したアドレスからはじまるマシン語サブルーチンを実行してもとの場所にもどってくるのだ。

ただ、USR関数はとにかく関数なので、BASICプログラムにおける現れ方は、ほかの関数とまったくおなじである。代表的な形式は、

A=USR(Ø)

という代入文の形で、いかにも、変数Aに引数(ひきすう)0で計算した関数USRの値を代入することが目的のように見える。しかし、たいていの場合、これはBASICの文法にいちおう従った儀式のようなもので、引数0にも変数Aにも意味のない場合が多い。

真の目的は「USR関数を使う」こと、それ自体にあり、DEFUSRでアドレスを指定しておいたマシン語プログラムを実行することだけが目的というケースが多いのだ。しかも、実行されるマシン語プログラムは、「A=〜」という計算とはまったく関係のないことをする。たとえば、画面の表示を禁止するとか、ふたたび表示するとか。

そういうことを実感するには、じっさいに使ってみるのがいちばんだ。マシン語プログラムもわざわざ作る必要はない。MSXにはBIOSというマシン語サブルーチン集がすでに入っているので、USR関数の定義をするだけで使えるものがいくつかある。

上の図に示している①〜④の手順どおりにダイレクトモードで打ちこんでみよう。①、②の段階ではべつになにも起こらないが、③でいきなり画面が消え、④でふたたび画面が現れる。

それがどうした、といわれるとても困る。「A=USRØ(Ø)」という計算式と画面の表示が禁止されるという現象との無関係さが美しいではないか。

サンプルプログラム解説

10 初期設定。マシン語をプログラム中に含む場合は、かならずこのようなCLEAR文が必要。ここでは&HDØØØ以降にマシン語プログラムを置くためにCLEAR文でメモリを確保し、そのアドレスにUSRØを定義

20 画面の初期設定。とくにこのプログラムではVRAMのアドレスが関係するためにSCREENØである必要がある。ここをSCREEN 1にする場合は、行40のデータの一部を変更する(行40の解説に指示)

30 マシン語プログラムを&HDØØØ以降に書きこむ

40 マシン語プログラムデータ。行20をSCREEN 1にしたい場合は、データ6個目の「CØ」を「ØØ」に、9、10個目の「ØØ, ØØ」を「ØØ, 18」に変更すること

50～80 USR関数呼び出し。引数になるA%は65から90まで変化しながらUSR関数を呼び出す。この場合、A%は画面を埋める文字のキャラクタコードになっている

※ちなみに、CTRL+STOPでこのプログラムをいったん中断したあと、U=USR(<好きなキャラクタコード>)をダイレクトに実行すれば、そのつど、好きな文字で一瞬にして画面を埋めることができる。

引数に意味のあるUSRの場合

前ページのUSR関数に使われた引数「0」は、じつは923でも71444でもなんでもかまわない。この場合は、ただひたすらBASICで扱える数値でありさえすればいいのだ。

それほどこうしたケースの引数には意味がないので、くだくだといわずに0を使うことが多い。さらに、USR0の0は省略できるため、たいていは省略してしまう。その結果、USR関数となると、

A=USR(Ø)

という形で見かけることが多く、USR関数を使うときは、カッコのなかに0を入れないといけないと思っている人がいたとしても笑えない。あはは。

こうした「無意味な引数」などのことを、プログラマ系の人々はダミー(dummy)と呼ぶ。引数にかぎらず、無意味な存在はダミーと呼ばれるようだ。ダミーとは、もともと物いわぬ人形や、デクノボーのことを意味する英語だが、どういうわけかいろんな形であちこちの分野で使われるので気をつけよう。

■引数に意味を持たせるには

ところで、USR関数の引数がダミーでないこともある。

下のプログラムは、その1つの例だ。行60で使われているUSR関数は、引数として入ってくる数値(ただし整数型にかぎる)をキャラクタコードとみなし、それに対応する文字で画面全体を一瞬にして埋める。あまりにそれが速いので、行70に時間待ちの空ループを入れないと下の4点の写真のような変化が目に見えないほどだ。

これは、前ページ同様、BIOSを利用したものだが、そのBIOSの機能は、

「HLレジスタで指定されるアドレス(VRAM)以降を、BCレジスタで指定される個数(バイト)だけ、Aレジスタにあるデータで埋める」

というややこしい機能であったため、その各種レジスタに適切な値をセットするためのマシン語を使わざるをえなかった。

この手のBIOSエントリを使うには、このようにどうしてもマシン語を避けてとおれない。また、USR関数の引数をたとえば「キャラクタコード」などのように意味を持たせるためにも、マシン語は必要だ。

■必要なマシン語のレベル

ただし、そのマシン語はたいへんかんたんなものでだいじょうぶ。ここで使ったマシン語に

■かんたんなマシン語を含むUSR関数のサンプル

```
10 CLEAR 200,&HD000:AD=&HD000:DEFUSR=AD
20 COLOR 15,4,7:SCREEN 0
30 FOR I=0 TO 13:READ A$:POKE AD+I,VAL("
&H"+A$):NEXT
40 DATA 21,F8,F7,7E,01,C0,03,21,00,00,CD
,56,00,C9
50 FOR A%=65 TO 90
60 U=USR(A%)
70 FOR W=0 TO 200:NEXT
80 NEXT:GOTO 50
```

↑アルファベットのAからはじめて次々に画面いっぱいに表示していく。行50で渡された引数は表示される文字のキャラクタコードだ

しても、
21　××　××
7E
Ø1　××　××
CD　××　××
C9
　の5種類で、それぞれ「HLレジスタに2バイトのデータを入れる」、「HLレジスタが示すアドレスからAレジスタにデータを持ってくる」、「BCレジスタに2バイトのデータを入れる」、「指定したアドレスにあるサブルーチン(BIOSなど)を呼ぶ」、「もときたところ(USR関数ではBASIC)にもどる」という意味。
　これに、あと10個くらいの命令を覚えていれば、いくつかのBIOSを使うためのマシン語プログラムくらいはすぐに作れるはずで、それについてはファンダムの「マシン語の気持ち」でやっていく予定だ。
　で、USR関数の引数をこのAレジスタに入れることもすぐにできる。
　USR関数の引数は、その数値の型によって扱われ方がちがうため、ここでは整数型に限定した。整数型の引数は、メモリのF7F8(16進数。以下、16進数を表す「&H」は省略)に入れられる(引数の値がFFを超える場合はF7F9に上位の桁を収める)。
　この数値を引っ張ってAレジスタに入れている部分がマシン語の「21, F8, F7, 7E」だ。以下、図のように次々にレジスタに決まった値がセットされ、BIOSが呼ばれて、VRAMのある領域が同一データで埋められる。ここでは、HLにSCREENØのパターン名称テーブルの先頭アドレスを入れたので同一キャラクタで画面中が埋められることになるわけだ。
　ここでは引数を利用するケースをあげたが、70ページには引数とその結果出てくるUSR関数の値の両方を利用するサンプルを掲載しているので興味のある人は参考にしてほしい。

■USR関数で呼ばれたマシン語の働き

①21, F8, F7, 7E

RAMのアドレスF7F8に、引数A%の値が収まっている。これをAレジスタに送りたい。そのために、「21、F8、F7」(HLレジスタにF7F8を入れる：マシン語では16進数4桁は2桁ずつ逆転させて置く)のあと、「7E」(そのHLレジスタが指すアドレスにある値をAレジスタに入れる)を実行してとりあえず引数をAレジスタに入れた。

②Ø1, CØ, Ø3

さて、このマシン語の目的はBIOSのØØ56にある「VRAMの指定領域を同一データで埋める」ためのエントリを呼ぶこと。ここは、Aレジスタにそのデータ、BCレジスタに書きこむ領域の長さを入れる必要がある。ここではSCREEN1の全画面ぶんの文字数、960(10進数)、16進数にしてØ3CØ。「Ø1、CØ、Ø3」は、BCレジスタにØ3CØを入れる命令だ。

③21, ØØ, ØØ

さらにそのエントリを呼ぶまえに、書きこみを開始するVRAMのアドレスをHLレジスタに入れれば準備完了だ。「21ØØØØ」とは、HLレジスタにØØとØØを書きこむ命令だ。

④CD, 56, ØØ

ここがBIOSを呼んでいるところ。「CD」は、次の2バイトで指定されるサブルーチン(この場合はBIOS)を呼ぶという意味。①〜③でパラメータ(A、BC、HLの各レジスタの値)をセットしたうえで一気にBIOSØØ56を呼べると思っていた。

⑤C9

この「C9」は、BASICのRETURNに似ている。USR関数で呼び出されるマシン語の最後に置かれるのがふつうで、これが実行されるとマシン語からBASICに帰ってきて、BASICリスト上、USR関数の次にあったステートメントに実行が移る。そのときに、F7F8にあった値をUSR関数の値として渡してしまうのだ。

RAMをちょっといじれば世界が変わる

BASICテクニック

#22 ワークエリアの不思議な働き

RAMの&HF380以降にMSXの聖域がある。めったなことでは触るべきではないといわれる、その聖域の名はワークエリア。でも、触っちゃおっと。

MSXの心の琴線「ワークエリア」

話の本題に入るまえに重要な注意をしておきたい。

フロッピーディスクを使っている人へ。

POKE文を使ってワークエリアをいじったりしたあとは、リセットするまでディスクを使わないでほしい。なにかのまちがいで、ディスクの中身がおかしくなってしまう可能性があるからだ。ただし、どんなことをしていても、いちどリセットすれば、またふつうにディスクを使ってかまわない。

■ワークエリアとは

さて、ワークエリアだ。

ワークエリアとは、そもそも「作業領域」という意味で、いろいろな種類がある。

たとえば、ファンダムなどでは多くの場合、「マシン語プログラムのワークエリア」という意味で出てくることが多い。その場合、そのマシン語プログラムが自機のX座標やY座標やマップデータなどをメモしておくために確保してある、RAMのある領域のことを指している。場合によっては、VRAMやディスクもそういう意味でのワークエリアになりうる。

しかし、今回の記事で扱うワークエリアは、こうしたユーザーが自分のプログラムのために使うワークエリアのことではなく、MSX自身がさまざまな目的で使うワークエリアを指す。

■システム・ワークエリア

基本的にRAMの&HF380から&HFFFFの領域は、コンピュータ・システムであるMSXがさまざまな作業をするために、きわめて〝プライベート〟に使っている。この作業領域が、「MSXのワークエリア」と呼ばれる部分だ。いわば、主婦のキッチン、サラリーマンのロッカー、医者のカバン、大工の道具箱、画家のパレット、作家のノート、中学生の机の引き出し、乙女の鍵付き日記のようなものだ(ちょっとちがうか)。

もう少し細かくいうと、&HF380～&HFD99の領域をとくにシステム・ワークエリアといい、画面の状態、キー入力の状態、BASICプログラムの実行状況などなどに関する興味深い部分がずらっとならんでいる。プログラム解説などでマシン語プログラムもないのに突然「ワークエリア」といった場合は、ここのことを指していることが多い。また、「BASICのワークエリア」という呼び方をする場合もある。

ともかく、ここはMSXの働きと深く関わりあっている核心部分で、いわばMSXの心の琴線。ちょっと触っただけでMSXはビビンと感じてしまう。だから、むやみにこのあたりをい

ワークエリア ディスクを接続したり搭載している機種では、ほかにもディスクのワークエリアが必要だ。ディスクのワークエリアは、公開されていないうえに機種、メーカーによってちがう。しかし、一般に&HDE40以降はディスクのワークエリアになっていると考えたほうがベストだといわれている。つまり、&HDE3Fまでがユーザーの自由に使えるメモリだというわけだ。

ものの本 「ワークエリア一覧」は、MSX用のマシン語入門書や54ページで紹介している本などに載っている。

ROM上の文字フォント格納アドレス じつは、このワークエリアは&HF91Fからの3バイトで、最初の1バイトには、ROMそのものの識別コードが入っているのだが、これについて触れると、話がもっとべつの方向に広がってわけがわからなくなってしまうので、ここでは実際のアドレス部分のみを紹介した。

ローマ字かな変換のモードスイッチetc. ここはMSX2以降の機能だ。明るい灰色の3ビットはMSX2+関係やJIS第2水準漢字ROMの有無を表しているがここでは省略。右端のビットを1にするとローマ字かな変換モードのオン／オフができるようだが、左端のひらがな／カタカナを表すビットは、1にしたり0にしたりしても、切り換わらない。ひらがな、あるいはカタカナを変換したあとに、ここのビットが書き換わる仕組みのようだ。

■ひと口カツふうワークエリアつまみ食い一覧

食べ方 赤い字がそれぞれのワークエリアのアドレスで、すべて16進表記(&H は省略)。大きな枠は1個4ビット(16進数)、小さな枠は1個1ビット(2進数)。なかに入っている数値は、それぞれの初期値(WAVY70F D2の場合)。POKEで書き換える場合は十分注意すること。

F3B0 1D
現在の画面の1行の幅
●現在の画面の1行に入る文字数。WIDTH文によっても変化する。ここを書き換えると奇妙な画面になる。

F3B1 18
現在の画面の行数
現在の画面での行数。ふつう24行だが、ここをたとえば25行にしたりするとカーソルが画面の下に隠れたりする。

F3DC 01
カーソルのY座標
●通常の座標系に比べて+1した数値が入っている。このワークを書き換えると、その数値に応じてカーソルが動く。

F3DD 01
カーソルのX座標
●通常の座標系に比べて+1した数値が入っている。このワークを書き換えると、その数値に応じてカーソルが動く。

F3E8 ジョイスティックのトリガーボタンの状態を保存

1	1	1	1	0	0	0	1
Ⓑ	Ⓐ	Ⓑ	Ⓐ				
ジョイスティックポート2		ジョイスティックポート1					スペースキー

●トリガーが押されると、それに対応するビットが0になる。たとえば、&B10010000(10進数144)だとスペースキー、ジョイスティック1のBボタン、スティック2のAボタンの入力を示す。

F920 BF
F921 1B
ROM上の文字フォント格納アドレス
●2バイトでROMの&H1BBFを指している(38ページの傍注参照)。このアドレス以降にキャラクタパターンデータがひとそろい入っていて、SCREEN文が実行されるたびにVRAMに転送される。ここを書き換えてSCREEN文を実行するとデータがずれ、文字の形が変になる。

FAFC ローマ字カナ変換のモードスイッチetc.

0	1	0	0	0	1	0	0

●このワークエリアはMSX2以降のみ有効。

●ひらがな/カタカナ
0=ひらがな
1=カタカナ

●VRAMアドレスのマスク
0=マスクする
1=マスクしない

●VRAMサイズ
00=16K VRAM
01=64K VRAM
10=128K VRAM

●ローマ字カナ変換
0=変換しない
1=変換する

FCA8 00
挿入モードのフラグ
●ここが0のときは通常の上書きモード。0以外のときに挿入モードになる。ただし、カーソルの形は変わらない。

FCA9 00
カーソル表示の有無
●動作中のカーソル表示の有無。0で表示なし、0以外は表示あり。LOCATE文の第3パラメータとおなじ。

FCAA 00
カーソルの形
●0のときは四角いカーソル。0以外のとき挿入モードとおなじカーソルの形。ただし、挿入モードにはならない。

FCAB 00
CAPSキーの状態
●ランプの表示ではなく、大文字モードかどうかを表す。0でオフの状態、255でオン(大文字モード)の状態。

FCAC 00
かなキーの状態
●これもランプ表示とは関係なく、かなモードかどうかを表す。0でオフの状態、255でオン(かなモード)の状態。

FCAD 40
かなキーの配列
●かなが50音順配列かJIS配列か。0で50音順配列。&H40(10進数64)でJIS配列。40ページの傍注参照。

じると、場合によっては、いきなり初期画面にもどったり、キー入力を受け付けなくなったりすることもある。

ものの本に載っている「ワークエリア一覧」を見ていると、ここをこうすればこんなことになるんじゃないか、とか、ここをいじったらいったいどうなるんだろう、とか、次から次に妄想がわいてきて、ドキドキしてしまう。禁断のイメージのために、かえって触りたくなってしまうのがワークエリアなのだ。

やりたいほうだいやってしまうのはいいことではないが、がまんしていても文明の進歩はない。どうせ、RAMだ。いざとなったらリセットすればもとにもどるのだ。

で、効果が見えやすく、かつ、わかりやすく、比較的無難そうなワークエリアのアドレスをいくつか選んでみた(上図)。

てきとうに数値を変えて遊んでいると、なんとなく、コンピュータという存在の性格がわかってくるような気がする。

現在の画面の１行の幅　このワークを通常のＷＩＤＴＨで設定できる数値以上にしてしまうと、もはや正常な動作は期待できない。たとえば、そうやってＬＩＳＴを実行すると、１行ぶんを表示するのに２－３行つかったり、あともどりしたり、画面の下に入りこんだりしてわけがわからなくなる。

50音順配列　ＭＳＸは当初、50音順配列が主流だったようだが、ディスクドライブ内蔵タイプのＭＳＸ２が主力になってきたころから、断然ＪＩＳ配列優勢になってきた。それもこれも、たぶん、入力しやすいローマ字かな入力機能がＭＳＸ２にあるため、50音順配列の必要がなくなったからだろう。50音順配列とＪＩＳ配列のキーボードではかなの位置ばかりでなく、句読点や半濁音などの位置もちがうが、ときには、古きよき50音順配列をＪＩＳキーボードによみがえらせるのも風流ではないか。

リスト１の解説

10　画面の初期設定。縦横比率を考えて、ＳＣＲＥＥＮ１にする

20　整数型宣言。変数ＡＤにキーマトリクスのワークの先頭番地を入れる

30　座標（０，５）にカーソルをセットし、ループをはじめる。ループはキーマトリクスのワーク11バイトぶんをすべて表示していくために０－10。

40　キーマトリクスのワークを１バイトずつ読んできて、それを２進表記で表示。ＰＥＥＫした値に256を加えているのは、９桁の２進数を作りたいため。９桁の２進数を作れば、その２桁目以降を表示することで確実に８桁の２進数の形になる。こうしないでそのまま読んできた値を２進数にしてしまうと、たとえば４は「１ØØ」などと３桁になって、キーマトリクスの表としてはふさわしくない。これは、ゼロサプレス（ゼロ抑制。よけいな０を表示しない）という機能なのだが、この場合は、その機能がじゃまなのだ。

50　ループ閉じ。行30からくりかえし

ワークエリア遊びのわびさび

■ＣＬＳなしに行の幅を変える

たとえば、&HF3BØの「現在の画面の１行の幅」というワークエリアなどは、ほぼ、ＷＩＤＴＨで設定された値が入ると考えていいだろう。

いろいろな幅をＷＩＤＴＨで設定しながら、

？　PEEK(&HF3BØ)

を実行してそのアドレスの数値を見れば、かならず一致する。

そればかりではなく、このワークエリアにてきとうな数値を書きこむと、その次の行から行の幅を変えてしまう。これはＷＩＤＴＨにはできない芸当だ。ＷＩＤＴＨだと、行の幅を変えるときにはかならずＣＬＳがかかるからだ。

それに、たとえば、ワークに直接書きこむ場合は、無法な値（たとえば50）をむりやり設定することさえできる。ただ、そうなると本格的に異様な画面になってしまう。

たとえば、

POKE　&HF3BØ，5Ø

を実行してみると、行の左端や右端の位置が中途半端なところにできて、とても奇妙だ。

この奇妙な現象はＳＣＲＥＥＮ文を実行すれば消える。

■キャラクタパターンの素

奇妙といえば、「ＲＯＭの文字フォント格納アドレス」というワークをいじるともっと奇妙なことが起こる。

文字フォントというのは、ようするにキャラクタパターンのことで、じつは、キャラクタパターンデータが１セット、ＢＡＳＩＣインタプリタなどが入っているＲＯＭのなかに置かれているのだ。ＳＣＲＥＥＮ文が実行されてテキスト画面が更新されるたびに、ＶＲＡＭのパターンジェネレータテーブルにＲＯＭからデータが送られてくる。そのとき、ＲＯＭのどこから引っ張ってくるかを記憶しているのがこのワークなのだ。

数値が2バイトにまたがっているときは、上位と下位を入れ換えてならべるので、&HF92Ø、&HF921の2バイトは、&H1BBFというアドレスからパターンデータが入っていることを意味する。そこで、&HF92Øにそこの値から8（バイト）を引いた値を書きこみ、ＳＣＲＥＥＮ文を実行すると、下の写真のようにわけのわからない画面になる。これは、すべての文字のパターンがキャラクタコードで１ずつずれているからだ。キャラクタコード表で確認してほしい。

■キーマトリクスの便利さ

ワークエリアのなかでも、右のページで紹介している「キーマトリクスの状態」は、たいへん便利なものだ。くわしくは右ページを読んでほしいが、これを利用すると、ＩＮＫＥＹ$では検知できないキー（ＳＨＩＦＴキーとかＧＲＡＰＨキーとか）の入力を調べることができる。

これと似ているのが、39ページに出ている&HF3E8のワーク「ジョイスティックのトリガーボタンの状態を保存」だ。

この2つのワークは、ＰＥＥＫ関数だけで使用するのでまったく危険性がない。

気をつけなければいけないのはＰＯＫＥを使う場合で、なにが起こるかは保証されていない。ファンダムなどに投稿するプログラムでは使わないように。

●さまざまに悩むカーソル

```
POKE &HF3B0,20
          Ok
          POKE &HF3DC,10



          Ok
          POKE &HF3DD,25                      Ok

          POKE &HFCAA,255
          Ok
```

↑ＳＣＲＥＥＮ０のＷＩＤＴＨ40の状態で、上から順に４つのステートメントをカーソル移動なしに実行していった画面。１行の幅が変わり、Ｙ座標、Ｘ座標が飛び、形が変わっている

●1字ずらしで気を変にしよう

```
POKE &HF920,PEEK(&HF920)-8:SCREEN0
```

↑上の命令文を実行してから、リスト１を表示し、「ＩＢＭ」と打ちこんだところ。うぎゃ

●ヌフアからアイウへ

```
ヌフアウエオヤユヨワ
Ok
POKE &HFCAD,0:POKE &HFCAC,255
Ok
アイウエオナニヌネノ
```

↑カナモードでキーの上段を左から右へ。次に命令文を実行したあとおなじキーを打った

BASICのマニュアルには載っていないキーマトリクス

color auto goto list run

◎リスト１の実行画面。ぜんぶ１なのは入力がないことを示す

color auto goto list run

◎テンキーとフルキーの１、Ｆ１、ＳＨＩＦＴ、Ａを同時に押した

アドレス								
FBE5	' 7 土 ャ ヤ	& 6 金 ォ オ	% 5 木 ェ エ	$ 4 水 ゥ ウ	# 3 火 ァ ア	" 2 月 フ	! 1 日 ヌ	0 万 ヲ ワ
FBE6	+ ; ♣ レ	{ [○ 「 °	@ ゛	| ¥ 円 ー	~ ^ へ	= - ー ホ	) 9 千 ョ ヨ	(8 百 ュ ユ
FBE7	B ┘ コ	A チ	_ ◆ ロ	? / ♠ ・ メ	> . 大 。 ル	< , 小 、 ネ	}] ● 」 ム	* : ♥ ケ
FBE8	J マ	I │ ニ	H 時 ク	G ┤ キ	F ┼ ハ	E ┌ ィ イ	D ├ シ	C └ ソ
FBE9	R ┬ ス	Q タ	P π セ	O ラ	N ミ	M 分 モ	L 中 リ	K ノ
FBEA	Z ッ ツ	Y 年 ン	X × サ	W テ	V ┴ ヒ	U ナ	T ┐ カ	S 秒 ト
FBEB	F8 F3	F7 F2	F6 F1	かな	CAPS LOCK	GRAPH	CTRL	SHIFT
FBEC	RETURN	SELECT	BS	STOP	TAB	ESC	F10 F5	F9 F4
FBED	→	↓	↑	←	DEL	INS	CLS HOME	SPACE

テンキー

アドレス								
FBEE	4	3	2	1	0	option	option	option
FBEF	.	,	−	9	8	7	6	5

キーマトリクスのワークエリアには、じつは新と旧の２種類ある。右上にでかでかと掲載しているのは、新のほうだ。新と旧の２種類があるのは、キーを押しっぱなしかどうかを判定するためらしいが、いずれにしても、ＢＡＳＩＣレベルでは関係なさそうで、新のほうだけを紹介した。じっさいにＢＡＳＩＣのプログラムで使われるのも、こちらのほうだ。

■キーマトリクス表の見方

現在は、ほとんどがＪＩＳ配列になっているようなので、あえてＪＩＳ配列のキーで示した。50音順配列の人は英数字で見てほしい。

赤い文字は39ページと同様、ワークエリアの16進表記のアドレスだ。それぞれのアドレスに入っている数値を２進表記したときの各桁に対応するキーが図になっている。押されていないときに１、押されると０になる。ふつうのオン／オフと逆なので注意。また、テンキーのなかにある「ＯＰＴＩＯＮ」(オプション)という部分はメーカーによって自由に使っていい部分で、機種によって異なる。

これで見ると、フルキー(かな文字といっしょになっているキー)の「１」と、テンキーの「１」とは、キーマトリクス上はちがうキーになっていることがわかる。

■キー入力の調べ方

上のキーマトリクスの図で、上からｎ番目(ｎはいちばん上を０として数える)、右からｍ番目(同様にいちばん右を０として数える)のキーを調べる場合は、

ＩＦ　ＰＥＥＫ(＆ＨＦＢＥ５＋ｎ)ＡＮＤ２＾ｍ＝ØＴＨＥＮ～

という条件文で調べられる。この場合、そのキーが押されていればＩＦ文のなかは成り立ち、～以下が実行される。

たとえば、ＧＲＡＰＨキーの場合、ｎは６、ｍは２なので、

ＩＦ　ＰＥＥＫ(＆ＨＦＢＥＢ)ＡＮＤ４＝ØＴＨＥＮ～

で調べられるわけだ。

■リスト1　キーマトリクスの状態をのぞく

```
10 COLOR 15,4,7:SCREEN 1:WIDTH 29
20 DEFINT A-Z:AD=&HFBE5
30 LOCATE 0,5:FOR I=0 TO 10
40 PRINT MID$(BIN$(PEEK(AD+I)+256),2)
50 NEXT:GOTO 30
```

思考パターンを図式化する伝統的技法

BASICピクニック

#24 たしなみとしてのフローチャート

フローチャートの技法は、ほんとうに役に立つのか。フローチャートを書くために必要十分な基礎知識と、実際にやってみた、かんたんな応用サンプル。

フローチャート　flowchart　流れ図。処理の流れを図式化したもの。JIS(日本工業規格)は、フローチャートに関して「JIS情報処理用流れ図記号(Flowchart Symbols and their Convention for Information Processing:いちおう英語の名前があったのでついでに書いておきました)」として30種類の記号および使い方を規定している。
「S←Ø」など、流れ図記号の内部に書いてある文字を「本文」という、など、いろいろと細かな規定があるが、MSXのBASICプログラム用のフローチャートとしては、今回の記事で掲載しているだけの知識で十分だろう。
スプレッドシート　spreadsheet　表計算ソフトのこと。たんにちょっとかっこをつけて、英語でいっているだけ。表計算ソフトのことを「スプレッドシート」というやつの7割は、そのつづりを知らない。
流れ線の交差　作図上、流れ線が交差せざるをえなくなることがあるが、交差したからといって、その2つの流れ線のあいだに論理的な関係があることにはならないので、気にせず、交差してかまわない。

フローチャートは99パーセントぶん

コンピュータが動いているとき、回路のなかでは、電気信号がクルクルと変化している。その意味では、熱血感動大涙のRPGにしろ、冷徹理知的血も涙もないSLGにしろ、ノンキなCGツールやとっぽいスプレッドシートにしろ、つきつめれば、おなじものである。どんなものであれ、ある瞬間を取り出せば、MSXのCPU(Z80A)は、たかだか50万ビットのメモリの電気信号のならびに従い、100万ビットのVRAMを介して、ディスプレイに映像信号を送り出しているにすぎない。

しかし、1ビット単位で見れば無意味でしかない、その変化が、何万、何十万と集まり、1秒間に何百万回も変化して、大きなパターンを形成すると、どこかで質的な飛躍がある。いつのまにか、それは、ブラウン管の向こうの、あたたかい意思を持った存在として感じられるようになるのだ。

それがソフトウェアと呼ばれるものだ。そして、ソフトウェアの実体であり、仮想的な意思として働くものがプログラムだ。

たとえば、BASICプログラムの場合、1つ1つのステートメントは、宣言したり、計算したり、表示したり、飛んだり跳ねたりしているだけで、とくに「考えている」わけではない。しかし、にもかかわらず、プログラムは全体として1つの「思考パターン」を成している。

プログラムのどこに、思考パターンが宿っているのか。それは、ステートメントの実行の順序、つまり手順にある。

XとYという座標変数があったとすると、①この変数の内容を更新し、②そのあとで座標(X，Y)に表示する、という手順と、①まず座標(X，Y)に表示し、②そのあとでX、Yの内容を更新する、という手順とでは、イラクとイランくらいのちがいがある。このちがいに、思考パターンが宿るのだ。

しかし、生のプログラムのままでは、そのもっともかんじんな思考パターンが見えにくい。

そこで、手順の流れを図式化して、思考パターン全体を見えやすくするための代表的な技法がフローチャートなのである。

その記号や書き方がJIS規格で定められているが、規格の30種のうちから必要十分と思われる12種を抜粋し、BASICにそくした解説をつけた。この表と、今回のサンプルを参考にすれば、あなたもすぐにフローチャートが書けるはずだ。

書いてみれば、すぐにわかる。フローチャートを書けば、プログラムは99パーセントできたもおなじだ、と。

5月号の訂正　5月号の『BASICピクニック』リスト1の解説中、「使い方」の項目で、ロード／セーブは「GOTO13Ø」を実行、と書いてありますが、「GOTO12Ø」の誤りです。訂正します。指摘してくださった方々、ありがとうございました。エラーを出した方々、どうもすいません。

■代表的な流れ図記号一覧

※「JIS情報処理用流れ図記号」より12種抜粋。薄赤の地が敷いてある記号だけでも実用的には十分。

記号	名前と意味
（長方形）	**処理** process あらゆる種類の処理機能を表す。つまり、オールマイティの箱だ。実際問題としては、これと「判断」と「流れ線」だけでフローチャートは書ける。
（ひし形）	**判断** decision 条件に応じて、複数の道筋のうち、どれをとるかの判断を表す。BASICのIF文、ON~GOTOなどの分岐命令に対応する。箱のなかに条件文などを書く。
（六角形）	**準備** preparation 画面の初期設定、変数初期化などの準備的な処理を表す。ループ、とくにFOR~NEXTのはじまりなどにも使う。
（二重線付き長方形）	**定義済み処理** predefined process サブルーチンの呼び出しを表す。GOSUB文やUSR関数にあたる。箱のなかにサブルーチン名を書き、サブルーチン用の独立したフローチャートを別に書く。
（台形）	**手操作入力** manual input キーボードやジョイスティック、マウスなどによる入力を表す。プラスX・ターミネーターレーザーを撃つ、なんてのも、ここ。
（平行四辺形）	**入出力** input/output 入出力一般を表す。BASICでは、READ文によるデータ読み出しや、ディスクを使ったデータの読み書きなどがこれで表される。
（波形の下辺）	**書類** document 正式な定義では「書類を媒介とする入出力機能を表す」とあるが、MSXのBASICの世界では実質的にプリンタからの出力のこと。
（表示記号）	**表示** display おもに、ディスプレイへの出力を表す。この形は、見てのとおり、ブラウン管をかたどっているのだ。また、CAPSランプの点滅もこの「表示」にあたるだろう。
（直線）	**流れ線** flow line 各記号を結びつける。原則として、流れの方向は、左から右、上から下。この原則からはずれるときは矢印を付ける。また、流れ線どうしの交差・合流も可。
（円）	**結合子** connector 作図上の理由で流れをワープさせるときに使う。また、準備とあわせてループにも使う。 ■結合子の使用例 (ⓐ)
（角丸長方形）	**端子** terminal, interrupt プログラムの開始と終了や中断を表す。開始のほうには、そのプログラムの名前を書いたほうがわかりやすい。プログラムによっては終了がないこともある。
（点線と括弧）	**注釈** comment, annotation いわばフローチャート用のREM文。点線の左側に流れ図記号、右側にその説明を書く。 ■注釈の使用例（S←0 …… 合計値を入れる変数Sの初期化）

■フローチャートのサンプルとプログラム

※解法のフローチャートの記号中でひんぱんに使用されている矢印は、左の変数に右の数式を代入するという意味で「=」にあたる。また、「i」のみ小文字なのは、たんに数字の1などと見わけやすいようにするため。

問題	解法	BASICプログラム
1+2+3+……+10を計算する	開始	
	S←0 / i←0	10 S=0:I=0
	i←i+1	20 I=I+1
	S←S+i	30 S=S+I
	i<10 (yes→i←i+1 / no↓)	40 IF I<10 THEN 20
	Sを表示	50 PRINT S
	終了	

ならべかえ　ソート(sort) ということが多い。また、分類ともいう。昇順(小さい順)、降順(大きい順)、50音順、JISコード順、都道府県順など、一定の基準で複数のデータをならべかえること。ここでは、得点の大きい順にならべかえる例を示した。代表的なデータ処理なので、50音順(この場合は、基本的にはキャラクタコードの大小関係が判断の中心になるだろう)などの手順も各自でやってみると、なにかの役にたつだろう。

アルゴリズム　algorithm　解法。前ページのフローチャートの上に「解法」と書いてあるが、アルゴリズムを図にしたものが、フローチャートになるのだ。アルゴリズムをもっとくだいていうと、「解き方」といっていいだろう。たとえば、「ツルがa羽、カメがb匹いて、a＋bはn、しかも足はm本である。a、bをn、mで表せ」などという数学の問題があったとすると、とりあえず連立方程式にして、それでbを消して、aを消して……という解き方の道筋がある。この道筋がアルゴリズムであり、図にしたものがフローチャートなのだ。

フローチャートを書くメリット

ところで、ほんとうにみんなはプログラムを組むまえにフローチャートを書いているのだろうか。そうでもない。

じつは、担当者のわたしは、これまでに3回しかフローチャートを書いたことがない。しかも、そのうち、2回は今回の記事のためだ。ふだん短いプログラムしか組む必要がないからだが、残りの1回は、ほんとうに必要に迫られてのことだった。

BASICピクニックでGML(グラフィック・マクロ・ランゲージ：DRAW文データ)をテーマにしたとき、記事を書くときに楽なようにと、GMLエディタを作ったときだった。

てきとうに線を引いてリターンキーを押すと、その図形に対応したGMLをベラベラベラと表示してくれるていどのやつでよかった。で、いきなり、プログラミングしはじめたのだが、すぐに頭が混乱して、よくわからなくなった。挫折し、思い直して、また挫折した。

そこで、ようやく、設計図を書く気になり、ノートとボールペンを持って、わたしは喫茶店に入った。そこで、ああしてこうしてと考えつつ、処理と判断しかない、原始的なフローチャートを書いてみた。B6の小さなノート5ページに渡ったが、GMLエディタの全体像を図式化するのにかかった時間は意外に少なく、1時間ていどだった。紙の上で、しかも、図式化しながら書いていくと、頭のなかが妙に整理されるのだ。

そのフローチャートにそって、プログラミングを再開すると30分くらいで、ほとんどできあがってしまった。

フローチャートのメリットは、処理の流れが、明確に見えてくることだ。だから、書いているうちに、自分が作りたいプログラムの構造がどんどんはっきりして、全体を把握しやすい。構造的なバグも見えやすい。

で、「ならべかえ」のプログラムをフローチャートから作ってみた(右ページ)。このフローチャートを見ながら実際にプログラミングしてみると、その効果が体験できるはずだ。改造もしやすくなる。アルゴリズムが形になって見えているからだ。

フローチャートは、解析にも役立つ。お試しあれ。

```
HOW MANY MEMBERS? 12
 1 NAME,POINTS? NOP,107
 2 NAME,POINTS? MOROP,203
 3 NAME,POINTS? CHIE,-128
 4 NAME,POINTS? ACKEY,-505
 5 NAME,POINTS? FUZZY,5
 6 NAME,POINTS? ATS,361
 7 NAME,POINTS? NORIPEE,0
 8 NAME,POINTS? F,-18
 9 NAME,POINTS? UNZU,99
 10 NAME,POINTS? MANBOW,21
 11 NAME,POINTS? MAGICIAN,-130
 12 NAME,POINTS? OKADA,-15
```

```
 1 ATS          361
 2 MOROP        203
 3 NOP          107
 4 UNZU         99
 5 MANBOW       21
 6 FUZZY        5
 7 NORIPEE      0
 8 OKADA       -15
 9 F           -18
 10 CHIE       -128
 11 MAGICIAN   -130
 12 ACKEY      -505
Ok
```

↑33ページのリスト2の実行画面。最初に人数(12)を入力し、名前、得点の順に入力。12人ぶんが終わると自動的にならべかえて、得点の高い順に表示している

■サブルーチンの書き方

「定義済み処理」の記号のなかには、サブルーチンの名前だけを書き、サブルーチンだけのフローチャートを別に書いておく。この場合、サブルーチン用フローチャートの端子には、サブルーチン名とRETURNの文字が書かれる。

■2つ以上の分岐

「判断」から出る分岐が、3つのときは、図左のように左、下、右から流れ線を出せばいいが、それを超える場合、図右のようにクシ型の流れ線にするのが一般的。この例のように右に出ていかず、下向きに出ていってもかまわない。

「ならべかえ」のフローチャートとプログラム

開始

画面の初期設定を入れよう(❶)

i＝0〜9

数を自由に設定したい(❷)

入力するときに番号を表示させたい(❸)

A$(i), A(i)

A$(i)：名前
A(i)：得点

i

i＝0〜8 ❷

j＝i〜9 ❷

A(i)<A(j)　no　yes

A$(i)とA$(j)
A(i)とA(j)を交換

j

i

i＝0〜9 ❷

(i+1);A$(i);A(i)

i

終了

リスト1からリスト2へ

リスト1、2は、名前と得点を一定人数ぶん入力すると、得点の高い順にならべかえるプログラムだ。まずフローチャートを書いて、リスト1をプログラミングし、次にフローチャートをチェックしながらリスト2へとバージョンアップした。

■変更点①＝画面の初期設定はまったく独立した部分なので、たんに行5を追加しただけ。

■変更点②＝最初は話を単純化するために10個のデータにかぎっていたが、どう考えても実用的でないのでN人に改造。フローチャートの最初のループに入るまえにNを決定(行6〜8)し、あとは、ループに関わる部分をNに置き換えていった。

■変更点③＝データ入力時に番号を表示するように改造(行10)。図は変えてないが、準備と手操作入力のあいだに「番号表示」処理を挿入した形。

■リスト1　最初のフローチャートをプログラム化したもの

```
10 FOR I=0 TO 9
20 INPUT "NAME,POINTS";A$(I),A(I)
30 NEXT
40 FOR I=0 TO 8
50 FOR J=I TO 9
60 IF A(I)<A(J) THEN SWAP A(I),A(J):SWAP
 A$(I),A$(J)
70 NEXT J
80 NEXT I
90 FOR I=0 TO 9
100 PRINT I+1;A$(I),A(I)
110 NEXT
```

■リスト2　反省しつつ修正を加えたもの

```
5 COLOR 15,1,1:SCREEN 0:WIDTH 40:KEYOFF    ❶
6 INPUT "HOW MANY MEMBERS";N
7 N=N-1
8 DIM A$(N),A(N)
10 FOR I=0 TO N:PRINT I+1;    ❸
20 INPUT "NAME,POINTS";A$(I),A(I)
30 NEXT    ❷
40 FOR I=0 TO N-1
50 FOR J=I TO N
60 IF A(I)<A(J) THEN SWAP A(I),A(J):SWAP
 A$(I),A$(J)
70 NEXT J
80 NEXT I
85 CLS    表示を整えるための追加
90 FOR I=0 TO N    ❷
100 PRINT I+1;A$(I),A(I)
110 NEXT
```

FM音源用BASICでもっと遊ぼう!

BASICテクニック

♯25　モーツァルトとFM音源

時間と根気とお金と才能とチャンスがなく、演奏家になれなかった人のためのFM音源ミュージック。FM音源用BASICでだれもが演奏家になれる。

楽譜と音名とオクターブの関係

楽譜は、それが持っている情報の豊かな可能性に比べて、きわめて安価なメディアだと思う。その楽譜の価値を引き出すことが、すなわち、演奏だ。

没後200年でなにかと話題になることの多いモーツァルトがおもしろい楽譜*1を残している。いっけん、1人の奏者用の楽譜だが、じつは二重奏用の楽譜、というものだ。2人目は、なんと反対側からのぞきこむ。右ページの楽譜は、その奇蹟的冗談の楽譜の一部だ。今回は、この楽譜を「演奏」してみよう。

まず、右ページ下のようにフレーズをMML*2化していく。

音名をひろうのは、たんにドレミをCDEに置き換えていどのことだから、ガスレンジでお湯をわかすよりもかんたんだ。

音の長さの指定を入れるのも、2分音符に「2」、8分音符に「8」と入れていくだけの単純な作業だから、急須にお茶の葉を入れるよりやさしい。

オクターブ情報も、下図のようにゾーンを色分けしておいて、ゾーンからはみだすたびに上か下かを指定すればいいのだから、熱いお湯を急須に注いで、しかるのちに湯飲みにお茶を注ぐ作業に比べればなんでもない。

この3段階で、メロディのMMLは完成する。お茶をいれるよりかんたんなわけだ。

＊1　おもしろい楽譜　モーツァルトの『バイオリン・デュエット』。31ページには、原曲のなかほど、約5分の1にあたる部分を掲載した。芥川也寸志著『音楽の基礎』(岩波新書E57／520円)の45ページに「モーツァルトの冗談」として掲載されている『Mozart: Violin Duet』に基づいた。

＊2　MML　Music Macro Language(ミュージック・マクロ・ランゲージ)の略。演奏データ用の言語。

＊3　「>」「<」　「>」は1つ上のオクターブへ、「<」は1つ下のオクターブへ移動することを示すMML。これを使うと、はじめの音の高さを変えるだけで全体の高さを変えられる。

＊4　Qコマンド　1つの音が「あ————————」と鳴るとすればその状態がQ8。それに対してQ3なら「あ——」、Q6なら「あ——————」。音の歯切れの表現などに使われる。

■図1　楽譜と音名とオクターブの関係

ピンクの地をひいてある部分がMMLでいうO4(オクターブ指定の初期値)の範囲だ。

※この記事で扱う「FM音源」は、FMパックまたは、MSX2+以降の機種(松下電器のFS-A1FXを除く)に内蔵されているMSX-MUSICを指します。FMパックを増設していないMSX2、MSX1では今回の記事のプログラムは使用できません。ただし、MSX-AUDIOがあれば、「CALL MUSIC」を「CALL AUDIO」に置き換えれば使用できます。

モーツァルトの冗談をFM音源で聴こう

モーツァルト『バイオリン・デュエット』(抜粋)

この楽譜はふつうに(正立)見るとフレーズA－0～3が読め、逆さに(倒立)見ると、フレーズB－0～3が読み取れる。フレーズA－0(正立の最初の1拍＋4小節)についてMML化のプロセスを具体的に追ってみた。

フレーズB-3

フレーズA-0　B　A　G　F+　G　R　R　D　E C G　C E　D B G　B D

フレーズB-2

フレーズA-1　C C A C B B G B　A　B　C C A C B B G B　AF+ A A A

フレーズB-1

フレーズA-2　E　E E G E　D B D D C A C C　D D G D E E G E　D B D D C A C C

フレーズB-0

フレーズA-3　B R R　G B D　G B D　R R R　R R R　F+ A D　F+ A D　R R R　R R R　G B D　G B D　R R R　C　B　A　G

フレーズA-0をMML化していくプロセス

❶まず、音名だけをひろう。小節線のかわりにスペースを入れておくとわかりやすい。

❷基本を4分音符として、ここでは2分音符と8分音符だけ長さの情報を追加。これでリズム部分ができた。

❸最初のBはO5。メロディはとちゅうでO4とO5を行き来している。この情報を「>」「<」[*3]を用いて入れる。これでメロディとしてのMMLは完成。

❹文字の節約。3小節目、4小節目にある「8」がうるさいのでLコマンドで8分音符の指定をまとめる。

❺音の装飾。Qコマンド[*4](音の割合)とVコマンド(音量)を組み合わせて、スラー(2音をなめらかに演奏する)の表現を追加してみる。

B	AGF+	GRRD	ECGCE	DBGBD	❶メロディの音名だけをひろう
B	AG2F+	GRRD	E8C8G2C8E8	D8B8G2B8D8	❷音の長さの情報を追加(4分音符以外)
>B	AG2F+	GRRD	E&C8<G2>C8E8	D8<B8G2B8>D8	❸オクターブ情報を追加(はじめをO4として)
>B	AG2F+	GRRD	L8EC<G2>CE	D<BG2B>D	❹8分音符の指定を整理
>B	AG2F+	GRRD	L8Q8V15EQ6V12C<G2>Q8V15CQ6V12E	Q8V15DQ6V12<BG2Q8V15BQ6V12>D	❺スラーの表現を追加(はじめはQ6V12)

MMLの修飾と音色について

メロディがMMLになったら、「PLAY#2,」で演奏するわけだが、演奏するときにはっきり意識する必要のあるのが「チャンネル」だ。

FM音源の演奏プログラムのはじめにかならずある、
CALL MUSIC～
という命令は、FM音源用BASICを使用する宣言と同時に、演奏のチャンネル構成を宣言しているのだ。

チャンネル構成とは、「PLAY#2,」のあとにカンマで区切られて続く演奏データ*1の、何番目がどういう音を受け持つかということだが、CALL MUSIC命令では、次の2つのことしか決めていない。

①リズム音を使うか使わないか
②FM音をいくつ使うか

CALL MUSICのカッコ内の数値で、最初が①のリズム音の有無、2番目がダミーの0、3番目以降が②のFM音をいくつ使うかを指定する(図2)。リズム音が「ある(1)」場合は、一般のFM音の最後のチャンネルの次がリズム音のチャンネルになり、リズム音も含めてすべてのFM音の最後のチャンネルの次からがPSGのチャンネルになる。

じつは、これは、メロディのMMLを作るときから関係している。今回の例にあげたモーツァルトの『バイオリン・デュエット』は、フレーズA－3(逆から見るとB－0)で同時に3音鳴る部分があるため、片側3チャンネルぶん、ABあわせて6チャンネル必要だった。MMLももちろん、6チャンネルぶん用意する。

メロディデータが完成し、チャンネルのことがわかったら、それだけでピアノ音による演奏ができる。音色の指定をしなければかってに音色番号0の「ピアノ1」の音で鳴るからだ。

しかし、せっかくFM音源なんだから、音色を楽しみたい。音色の指定専用の命令(CALL VOICE)もあるが、1つ1つのチャンネルで「@〈音色番号〉」として音色を指定したほうがわかりやすい。ただ、左下の表のように、同時には使えない音色グループがあるという点に十分注意してほしい。

ここまでの知識で、FM音源による演奏は十分に楽しめるはずだ。あとは、QコマンドやVコマンドなどをこまめに入れたり(図3参照)、音を重ねたりして*2表情をつけるくふうをするわけだが、このへんは、実際に演奏させながら、試行錯誤で研究していくしかない。

*1　カンマで区切られて続く演奏データ　データがならんだ順に演奏されていくのだと思っている人がいるらしい。編集部には2人もいた。あなたも、そう思っていませんでしたか？

*2　音を重ねたりして　おなじメロディを異なる音色で同時に演奏すると、音色の合成に近い効果が得られる。音色ごとに音量も調整したり、意識的にずらしたりするのも効果的。

サンプルプログラムの解説　●行20「DEFSTR　A－C」はA、B、Cで始まる変数を「$」なしに文字変数として使うための命令。ふつうは使わないほうがいいが、音楽プログラムではほとんどが文字変数なので効果的に使える。●行40　フレーズA－3、B－0での和音用のチャンネルのMMLを設定。3、4行目でFM音源の各チャンネル初期化用の変数Cを設定して初期化。「O4L4」は初期値とおなじなので省略してもいい。音色は曲の構造のわかりやすいギター(10番)にしてみたが、どうだろうか。5行目の頭の「10」をてきとうに変えれば、いろんな音色での演奏が聴ける。●行60　これ以降が、おもな演奏データ。31ページのフレーズ番号をデータのわきに赤で書いてある。

■図2 CALL MUSICで設定されるチャンネル

●バイオリン・デュエットのプログラムサンプルの場合

```
CALL MUSIC(0,0,1,1,1,1,1,1)  残りは無条件でPSG用に設定される
PLAY#2,FM1,FM2,FM3,FM4,FM5,FM6,PSG1,PSG2,PSG3
```

ここが0のときリズム音なし、1のときリズム音あり

●もっとも基本的な初期化　⇧カンマで区切られて続く演奏データはすべて同時に演奏される

```
CALL MUSIC = CALL MUSIC (1,0,1,1,1)
PLAY#2,FM1,FM2,FM3,RYTHM,PSG1,PSG2,PSG3
```

■同時に使える音色グループ

@n	音色名
0	ピアノ1
2	バイオリン
3	フルート1
4	クラリネット
5	オーボエ
6	トランペット
9	オルガン
10	ギター
12	エレキベース
14	ハープシコード1
16	ビブラフォン
23	シンセ・ベース
24	シンセサイザー
33	エレクトリック・ピアノ2
48	金管楽器

「同時に使える音色グループ」は、比較的単純な音が多く、1つの演奏のなかで無制限に使えるが、魅力的な音や不思議な音が多い「同時には1種類しか使えない音色グループ」は、同時に鳴る音の中ではこのうちの1つしか使用できない。この禁を破ると、最後のチャンネルで指定した音色番号のみが有効となり、あとは無視される。

■同時には1種類しか使えない音色グループ

@n	音色名
1	ピアノ2
7	パイプオルガン1
8	シロフォン
11	サンツール1
13	クラビコード1
15	ハープシコード2
17	琴1
18	太鼓
19	エンジン1
20	UFO
21	シンセサイザベル
22	チャイム
25	シンセ・ドラム
26	シンセ・リズム
27	ハーモ・ドラム
28	カウベル
29	クローズ・ハイハット
30	スネア・ドラム
31	ベース・ドラム
32	ピアノ3
34	サンツール2
35	ブラス
36	フルート2
37	クラビコード2
38	クラビコード3
39	琴2
40	パイプオルガン2
41	PohdsPLA
42	PohdsPRA
43	チャーチオルガンL
44	チャーチオルガンR
45	シンセ・バイオリン
46	シンセ・オルガン
47	シンセ・ブラス
49	三味線
50	マジカル
51	フワワ
52	ワンダーフラット
53	ハードロック
54	マシーン
55	マシーンV
56	コミック
57	SE-コミック
58	SE-レーザー
59	SE-ノイズ
60	SE-星1
61	SE-星2
62	エンジン2
63	無音(ユーザーのオリジナル音色用の空きエリア)

■図3 スタッカートとスラーの表現例

音を切るスタッカートはQの指定値を小さくすればいいのでかんたん。2つの音をなめらかに演奏するスラーの表現では、Qを最大にしたうえに音量も多少いじってみた。

```
(L8)CCAC<BB>G<B
```

⇦基本的なMML

⇩修飾されたMML

```
Q3CCQ8V15AQ6V12CQ3<BBQ8V15>GQ6V12<B
```

バイオリン・デュエットの演奏プログラムサンプル

```
10 CALL MUSIC(0,0,1,1,1,1,1,1)
20 DEFSTR A-C:DIM A(3),A1(3),A2(3),B(3),
B1(3),B2(3)
30 FOR I=0 TO 3:READ A(I),B(I):NEXT
40 B1(0)="R R1 RBBR R>CCR R<BB":B2(0)="R
 R1 RDDR REER RDD":A1(3)="RBBR RAAR RBB"
:A2(3)="RDDR RDDR RDD":C="O4L4V12T160Q6@
10":PLAY#2,C,C,C,C,C,C
50 FOR I=0 TO 3:PLAY#2,A(I),A2(I),A3(I),
B(I),B2(I),B3(I):NEXT
60 DATA >B AG2F+ GRRD L8Q8V15EQ6V12C<G2>
Q8V15CQ6V12E Q8V15DQ6V12<BG2Q8V15BQ6V12>
D                                    ── A(0)
70 DATA >D C<B2A R>GGR RGGR RGG<B    ── B(0)
80 DATA Q3CCQ8V15AQ6V12CQ3<BBQ8V15>GQ6V1
2<B A2B2 Q3>CCQ8V15AQ6V12CQ3<BBQ8V15>GQ6
V12<B Q8V15AQ6V12F+Q3AAQ6A2          ── A(1)
90 DATA L8Q3AAQ6>C<AQ3GGQ6BA F+DQ3F+F+Q6
GDQ3GG AAQ6>C<AQ3GGQ6BG F+DQ3F+F+Q6F+2 ── B(1)
100 DATA >E2Q3EEQ8V15GQ6V12E Q8V15DQ6V12
<BQ3>DDQ8V15CQ6V12<AQ3>CC DDQ8V15GQ6V12D
Q3EEQ8V15GQ6V12E Q8V15DQ6V12<BQ3>DDQ8V15
CQ6V12<AQ3>CCQ6V12                   ── A(2)
110 DATA >C2Q3CCQ6EC <BDQ3BBQ6ACQ3AAQ6 B
2>C2 <BGQ3BBQ6ACQ3AAQ6               ── B(2)
120 DATA L4<B>GGR RF+F+R RGGR C<B2A G ── A(3)
130 DATA GB>D2<BG F+A>D2<AF+ L4GRRD ED2C
 <B                                  ── B(3)
```

リスト中の青い文字はスタッカートとスラーの表現用なので入力しなくてもメロディは聴くことができます

確認用データ

使い方は67ページ

10>tv20	20>wdA0	30>W850	40>Pwk2
50>2aJ0	60>rCo0	70>dp60	80>VwT1
90>cHj0	100>eKY2	110>67R0	120>Qr70
130>c4B0			

オリジナル合成音を作りたい人に

BASICピクニック

#26 FM音源の音色合成をめぐって

前回に引き続き、今回もFM音源について。魅惑の音色作りの仕組みから、音色データの構造を探り、音色の視覚化に挑戦してみた。

＊1 FM FrequencyModulation(フリーケンシー・モジュレーション：周波数変調)の略。ことばはむずかしいが、音の場合、音の高さを微妙に揺れ動かすことになる。揺れがゆっくりしている場合は、ビブラートになるが、それが高速になると、波形そのものが変質し、新しい音色に変化していく。ちなみに、このFM合成方式を発明したのは、アメリカのジョン・チョウニング博士。

＊2 ジョセフ・フーリエ Joseph Fourier 19世紀フランスの数学者・物理学者。数学的にいうとすべての周期的関数は、サイン関数の組み合わせに変換できることを証明した。この変換のことをフーリエ変換という。ちなみに、人間の三半規管は、音をフーリエ変換しているらしい。

＊3 蟹のカノン フリードリッヒ大王に与えられた主題をもとにした、J.S.バッハ『音楽の捧げもの』に収められているカノン。一般に、主題を逆から演奏していくカノンを「蟹カノン」というらしい。このあたりの話についてくわしくは、『ゲーデル、エッシャー、バッハ／あるいは不思議の環』(R・ホフスタッター著／白揚社)にコンピュータや数学の話とおりまぜながら書かれている。

＊4 OPLL ヤマハの開発したFM音源チップでMSX－MUSICに採用されているもの。

FM音源の音色はどこから来たのか

そもそもFM[*1]音源とは、なんなのか。ラジオのFMとFM音源は、直接の関係はないが、この2つのFMは、どちらもFrequency Modulation(フリーケンシーモジュレーション)の略で、日本語に訳せば「周波数変調」。ある意味で、おなじ方式なのだ。

周波数とは、周期的な振動が1秒間にくりかえされる数のことで、単位はヘルツ(Hz)。

音叉(おんさ)の音は、基準音のラ(A)で、周波数は440ヘルツ、これは、音叉が1秒間に440回振動することを意味する。だから、音叉のまわりの空気も440ヘルツで振動し、空気を介してラの音がまわりに広がっていく。

この周波数を時間的に細かく変化させるとさまざまな音色が出てくる、というのがFM音源の考え方なのだ。

サイン関数が合成されて大きな方形波に

■サイン波の合成

音色の母は、波形である。

さまざまな音はさまざまな特徴のある波形を持っている。波形はすなわち音源となるものの振動のパターンだ。極端な話、オシロスコープで見られるような波形とまったくおなじパターンでなにかを震えさせることができれば、そこからはバイオリンやフルートやハープの音などあらゆる音が出てくることになる。しかし、現実にはそんなことは無理だ。

ところが、どんな複雑な波形

リスト1：サイン波を合成して方形波を作る

```
10 COLOR 15,0,0:SCREEN 7:DEFINT A-X
20 D#=ATN(1)*4/128:V=80:SY=100
30 LINE (511,100)-(0,100),8
40 FOR X=1 TO 511:Y=0
50 FOR I=0 TO 2:P=2*I+1:Z=V/P*SIN(X*D#*P
):PSET(X,SY-Z),4:Y=Y+Z:NEXT
60 PSET(X,SY-Y):NEXT
70 GOTO 70
```

リスト1のプログラム解説

●行50が重要。変数Zは、中心線(Y座標100の水平線)からの距離をドットで表し、それぞれの単純なサイン波の振幅(強さ)を表す。Iのループのなかで、毎回、大中小の3つのサイン波をたどるドットをかくのに使われつつ、合成波をかくための変数Yに次々に加えられていく。

※今回の記事のプログラムを実行するには、MSX2(VRAM128K)以降でMSX-MUSIC(またはMSX-AUDIO)が必要です。

リスト2：音色を重ねてバッハを演奏してみる

```
10 CLEAR 700:CALL MUSIC(0,0,1,1,1,1,1)
20 CALL TEMPER(4):DEFINT A-Z
30 A$="{CE-}{GA-}{<B&BR>G&}{GG-&G-F&}{FE
&EE-&}{E-DD-C}{<BG>CF}{E-D}{CE-}{GFG>C<G
E-DE-}{FGA-B->C<E-FG}{A-DE-FGFE-D}{E-FGA
-B-A-GF}{GA-B->CD-<B-A-G}{AB>CDE-C<BA}{B
>CDE-FD<G>D}{CDE-FE-DC<B}{>C<GE-C}"
40 L=LEN(A$):FOR I=1 TO L:C$=MID$(A$,L+1
-I,1):C=ASC(C$):IF C>64 AND C<83 OR C=38
 THEN B$=B$+C$:NEXT ELSE IF C=45 THEN B$
=B$+MID$(A$,L-I,1)+C$:I=I+1:NEXT ELSE B$
=B$+CHR$(C+2*SGN(124-C)*SGN(61-C)*SGN(10
0-C)):NEXT
50 T$="T144L1":PLAY#2,T$,T$,T$,T$,T$:PLA
Y#2,"O5@14V15","O4@14V8","O5@14V12","O4@
2V8","O3@10V15"
60 PLAY#2,A$,A$,B$,B$,A$
70 SWAP A$,B$:GOTO 60
```

確認用データ 使い方は71ページ	10>hj40	20>Gk30	30>wJB6	40>s1s6
	50>Div0	60>0I20	70>6C10	

参考楽譜：バッハ『蟹のカノン』

リスト2のプログラム解説

●行10：5チャンネルのFM音をセット。●行20：なんとなく、音律にヴェルクマイスター(別)を採用。●行30：上の楽譜をふつうに見たMML。音の長さの指定をせずに「{」だけを使っているのは、左右どちらから見ても1つの音の長さがおなじになるようにするため。●行40：A$に入っているMMLを完全に逆の順序に並べてB$に入れる。「{」や「く」は逆向きのものに置き換える必要があるためいくつかのIF文がある。B$は、上の楽譜をお尻から見ていったメロディになる。●行50：各チャンネルの初期設定。第1～第3チャンネルにハープシコードを使い、第1、第3は1オクターブ高く設定。第4にピアノ2、第5にオクターブ低いギターを設定。●行60～行70：A$、B$の内容を交換しながら、なんどでも演奏。

も、基準となる周波数と、その整数倍の周波数のサイン波に分解できる(と、ジョセフ・フーリエ[*2]がいった)。逆にいえば、あらゆる波形は、複数のサイン波で合成できることになる。

その理論をちょっと実感するためのものが左ページのリスト1だ。リスト1を実行すると、青い点で大小のサイン波が描かれ、それらを単純に重ね合わせただけの波形を白い点で描く。この白い点の波形は、方形波といい、クラリネットなどに近い音色になるはずだ。

サイン波を人工的に出すのはかんたんなので、このフーリエの理論を応用すれば、さまざまな音色を作り出すことができることになるわけだ。

単純に音色を重ねあわせるだけでもけっこう奥深い響きが出る。リスト2は、16小節の楽譜を頭から演奏するラインと、お尻から演奏するラインが二重奏になっているというバッハの回文21面相的な『蟹(かに)のカノン』[*3](前回のモーツァルトの冗談と近いノリ)を音の重ね合わせで演奏してみたものだ。

■2つのオペレータ

こうした単純な重ね合わせを足し算とすると、FM方式の音色合成は、ダイナミックなかけ算のイメージだ。

MSX－MUSICに使われているLSI(OPLL[*4])のなかにはサイン波を発する、性格の異なるオペレータが2つ直列にならび(図1)、ダイナミックな波形を作り出している。モジュレータはキャリアの周波数を揺さぶりつづけていて、これが周波数変調(FM)なのだ。

リスト3は周波数変調によって波形がどう変わるかをシミュレートしたものだ。行80の、サイン関数のなかに入りこんでいるサイン関数が図1のモジュレータに相当する。

■図1 FM音源LSI「OPLL」の基本構造

モジュレータ
↓ 変調
キャリア
↓ 発音
音

リスト1に近い効果を出してみた

モジュレータのマルチプルを大きくすると

変調指数を大きくするとこんなに複雑に

リスト3：FM合成のシミュレーション

```
10 COLOR 15,0,0:SCREEN 5
20 OPEN "GRP:" AS #1:DIM M(15):FOR I=1 T
O 10:M(I)=I:NEXT:M(0)=.5:M(11)=10:M(12)=
12:M(13)=12:M(14)=15:M(15)=15:P#=4*ATN(1
):SY=100:SW=4/256*P#:A=50
30 MC=1:MM=2:IN=1.2 'MULTIPLE & INDEX
40 FC=SW*M(MC):FM=SW*M(MM)
50 DRAW"BM20,10":PRINT#1,USING"MC=##    M
M=##    IN=##.#";M(MC);M(MM);IN
60 LINE (255,100)-(0,100),7
70 FOR T=0 TO 255
80 F=A*SIN(FC*T+IN*SIN(FM*T))
90 LINE -(T,SY-F),15-C*2
100 NEXT
110 C=(C+1) MOD 7:GOTO 60
```

リスト3のプログラム解説

●行20：配列設定は、OPLLに使われる「マルチプル」というパラメータの変換テーブル。マルチプルとは、基準周波数に対する倍率(それはかならずしも音程を意味しない)のこと。●行30：MCは、キャリアのマルチプル、MMは、モジュレータのマルチプル。INは、変調指数というパラメータで、モジュレータの作用の大きさを表す。この3つの変数をいろいろ変えてやってみたのが左の写真。この大幅な変化こそが、FM音源のFM音源たるゆえん。●行80：内側のサイン関数がモジュレータで外側のサイン関数がキャリア。

PSGより複雑なエンベロープ

MSX−MUSIC(OPLL)では、キャリア(オペレータ1)が出すサイン波に、モジュレータ(オペレータ0)がサイン波で周波数変調をかけ、その結果、複雑な波形を生み出し、音色を合成している。

しかし、波形は音色の母にしかすぎない。じつは、音色には父もいるのである。

音色の父は、エンベロープという。PSGをあるていどやったことのある人なら、エンベロープということばにはなじみがあるだろう。例の「音量の時間的変化」というやつである。

PSGでは、8種類のエンベロープパターンしかなかったが、MSX−MUSICのエンベロープは、きわめて複雑で変化に富んでいるうえに、モジュレータ用とキャリア用にべつべつのエンベロープがある。

MSX−MUSICのエンベロープは、おもに4つのパラメータと、1つのスイッチで決定される。モジュレータにのみ、もう1つ全体の大きさを制限するパラメータもある。

4つのパラメータとは、頭文字を取って「ADSR」と呼ばれ、次のような意味を持つ。

1. AR(Attack Rate：アタックレイト) キーオン(発音開始)から、最大レベル(キャリアの場合は最大音量に相当する)に達するまでの時間計算用パラメータ

2. DR(Decay Rate：ディケイレイト) 最大レベルからサスティンレベルに達するまでの時間計算用パラメータ

3. SL(Sustain Level：サスティンレベル) 持続音タイプの場合は、キーオフ(発音終了)まで保持するレベル、減衰音タイプの場合は、DRからRRへの変化点のレベル。持続音か減衰音かは、図3の⑤中にあるEGというビットで決まる

4. RR(Release Rate：リリースレイト) サスティンレベルから減衰して0レベルに達するまでの時間計算用パラメータ。持続音の場合はキーオフから、減衰音の場合はサスティンレベルに達してすぐ。

このほかに、モジュレータにはTL(Total Level)というパラメータがあり、モジュレータの最大レベルを決めている。

また、SLの項で述べたように、図3の⑤のEGが持続音と減衰音の2種類を区別している。

エンベロープは、以上のパラメータだけで決まる。

リスト5は、MSX−MUSICの合成音色で設定されているエンベロープおよび各設定データを見るためのものだ。とりあえず、これで合成音色のようすを調べ、そのデータに手を加えながらイージーエディタ(リスト4)でオリジナル音を作ってみよう。できあがったら、またリスト5の行420〜450に移して、エンベロープのようすなどを観察すると、だんだん音作りのコツがわかってくる。

エンベロープ以外は、マルチプルが重要なパラメータだ。

■図2 OPLLで使われているエンベロープ

AR＝アタックレイト　DR＝ディケイレイト
RR＝リリースレイト　SL＝サスティンレベル
key on(キーオン)＝発音開始　key off(キーオフ)＝発音終了

キーオン、キーオフという表現には少しとまどうかもしれないが、音を出す、止めるを、鍵盤になぞらえていっているだけなのだ。この表現は通常使われているので覚えておこう。

キャリアのエンベロープの「強さ」は音量、モジュレータのエンベロープの「強さ」は変調量。変調量という概念はピンと来ないだろうが、音色変化の派手さといったところか。

■図3 リスト4のデータ構造

※各桁はすべて0〜Fの16進数になっている。

※各項目の意味は傍註参照。

■図3の項目説明

音色データは基本的に4×4、つまり、DIM A%(3, 3)で宣言される配列を使用した(整数宣言していない場合はかならず「%」が必要)。

①音色名　1つの配列要素にキャクタコードが2字ぶんずつ計8文字が入る。

②ボイス移調　トランスポーズ(TRANSPOSE)。その音色だけに適用される移調。10進数にして250が半音に相当する。

③フィードバック　モジュレータがモジュレータ自身を変調する割合。0〜15までだがビット0は意味がない。

④KSL、TL　この2バイトの上位2ビットがレベルキースケール(KSL)で、音程があがるほど出力レベルが下がる傾向を設定する。残りの14ビットがTL(トータルレベル)。TLは、モジュレータのみ有効で、数値が大きいほど最大レベルが低くなる。

⑤AM／PM／EG／KSR　各1ビットずつ。AM、PMはそれぞれトレモロ、ビブラートのオン／オフ(1でオン)。EGは0で減衰音、1で持続音。KSRは音程があがるほどエンベロープの変化が速くなる傾向の設定。

⑥マルチプル　基本周波数に対する倍率。前ページのリスト3行20を参照。

⑦エンベロープパターン　SLは小さいほどレベルが高く、RR、AR、DRもそれぞれ数値が小さいほど時間が長い。ただし、RR以降が0の場合は「変化しない」という意味になるので、ARを0にするとまったく変化しなくなる。また持続音に設定されたキャリアのRRを0にすると、音が鳴りやまなくなる。

リスト4のプログラム解説

●行10：FM音源のプログラムでは、かならずCALL MUSICを最初に持ってくること。この命令には「CLEAR」の機能も含まれているため、この命令のまえに設定した変数や配列宣言などはみんなもとにもどってしまうので注意。また、音色データを入れる配列Aは、かならず、整数型で4×4以上(または16以上)であること。●行20：音色名(行60のデータ。なくてもいい)の配列への読みこみ。この行は「READ A$」だけを残し、あとは入力しなくてもかまわない。●行30：行70〜行90のデータ読みこみ。●行40、50：配列のデータをユーザー用音色番号63にコピーして鳴らす。

リスト4：世界一イージーな音色合成エディタ

```
10 CALL MUSIC:DEFINT A-Z:DIM A(3,3)
20 READ A$:FOR M=0 TO 3:A(M,0)=ASC(MID$(
A$,2*M+1,1))+ASC(MID$(A$,2*M+2,1))*256:N
EXT
30 FOR M=1 TO 3:FOR N=0 TO 3:READ A$:A(N
,M)=VAL("&H"+A$):NEXT:NEXT
40 CALL VOICECOPY(A,@63)
50 PLAY#2,"@63V15O4A1"
60 DATA Piano 2         :'TONE NAME
70 DATA 0C00,0008,0000,0:'TRPS,FB
80 DATA 0F30,10D9,0030,0:'MODULATOR
90 DATA 0010,F3B2,0000,0:'CARRIER
```

【使い方】図3を参考に行60〜行90のデータを変更し、RUNするとオリジナル音が鳴る。とりあえず@1のピアノ2のデータを入れておいた。

リスト5：MSX-MUSIC内蔵の合成音色を見学する

```
10 '●●● FIRST SETTING
20 CALL MUSIC:OPEN"GRP:"AS#1:DEFINT A-Z:
DIM A(3,3),D$(3,3),N$(1),NG(14),PR(4)
30 DEFFNP(N,M,L)=VAL("&H"+MID$(RIGHT$("Ø
ØØ"+HEX$(A(N,M)),4),L,1)):PS#=1.5:DEFFNR
(I)=7Ø*(2/PS#)^-PR(I):DEFUSR=342
40 FOR I=1 TO 14:READ NG(I):NEXT:DATA 2,
3,4,5,6,9,1Ø,12,14,16,23,24,33,48
50 H=95:L=5:KO=1Ø:KF=19Ø:N$(Ø)="ﾓｼﾞｭﾚｰﾀ"
:N$(1)="ｷｬﾘｱ":LG#=LOG(1Ø)/2Ø/PS#
60 '●●● INPUT
70 COLOR 15,14,Ø:SCREEN Ø:WIDTH 4Ø:KEYOF
F:A=USR(Ø):INPUT"TONE NUMBER";TN#
80 IF TN#>Ø AND TN#<64 THEN TN=TN#:FOR I
=1 TO 14:IF TN<>NG(I) THEN NEXT:GOTO 1ØØ
90 BEEP:GOTO 7Ø
100 IF TN=63 THEN GOSUB 38Ø:CALL VOICECO
PY(A,@63) ELSE CALL VOICECOPY(@TN,A)
110 FOR M=Ø TO 3:FOR N=Ø TO 1:D$(N,M)=RI
GHT$("ØØØ"+HEX$(A(N,M)),4):NEXT:NEXT
120 TR$="ﾄﾗﾝｽﾎﾟｰｽﾞ="+D$(Ø,1):FB$="ﾌｨｰﾄﾞﾊ
ﾞｯｸ=    "+MID$(D$(1,1),4)
130 TN$=STR$(TN)+"=":FOR I=Ø TO 3:CH$=RI
GHT$("ØØØ"+HEX$(A(I,Ø)),4):TN$=TN$+CHR$(
VAL("&H"+MID$(CH$,3,2)))+CHR$(VAL("&H"+M
ID$(CH$,1,2))):NEXT
140 '●●● BASIC LINES & TONE NAME
150 SCREEN 5
160 LINE(KO,Ø)-(KO,H*2),8
170 LINE(KF,Ø)-(KF,H*2),7
180 FOR I=Ø TO 1:FOR J=Ø TO 1:LINE(Ø,H*(
J+1-I)+L*I)-STEP(255,Ø),I+1:NEXT:NEXT
190 DRAW"BM12,193":PRINT#1,TN$
200 '●●● DRAWING & PLAYING(44ØHz)
210 FOR OP=Ø TO 1:FL=H-L:GOSUB 33Ø
220 IF PR(3)=Ø THEN 29Ø '● PR(3)=AR
230 LINE (KO,H+H*OP)-STEP(AR,-FL)
240 IF PR(4)=Ø THEN LINE -STEP(KF-(KO+AR
),Ø):SL=Ø:RR=EG*RR:GOTO 26Ø '● PR(4)=DR
250 LINE -STEP(DR,SL):IF EG THEN LINE -S
TEP(KF-(KO+AR+DR),Ø)
260 DRAW"C9M+8,-6M-3,-2M-5,8" '● MARK
270 IF PR(2)=Ø THEN LINE -STEP(255-(KO+A
R+DR),Ø):GOTO 29Ø '● PR(2)=RR
280 LINE -STEP(RR,FL-SL)
290 NEXT
300 COLOR 1,Ø:FOR OP=Ø TO 1:PSET(196,11+
H*OP),Ø,TPSET:PRINT #1,N$(OP):FOR N=Ø TO
 1:DRAW"BM+196,Ø":PRINT #1,D$(N,OP+2):NE
XT:NEXT:DRAW"BM116,193":PRINT #1,TR$:DRA
W"BM116,2Ø1":PRINT #1,FB$:COLOR 15,14
310 PLAY#2,"@=TN;V15O4L1A"
320 IF STRIG(Ø) THEN 7Ø ELSE 32Ø
330 '●●● PARAMETER READING SUB
340 M=OP+2:FOR I=1 TO 4:PR(I)=FNP(1,M,I)
:NEXT:AR=FNR(3):DR=FNR(4):RR=FNR(2):EG=S
GN(FNP(Ø,M,3) AND 2)
350 IF OP=Ø THEN TL=(FNP(Ø,2,1)*16+FNP(Ø
,2,2)) AND 63:FL=FL*EXP(-TL*.75*LG#)
360 SL=FL*(1-EXP(-PR(1)*3*LG#))
370 RETURN
380 '●●● ORIGINAL TONE DATA SETTING
390 RESTORE 42Ø
400 READ A$:A$=LEFT$(A$+SPACE$(8),8):FOR
 M=Ø TO 3:A(M,Ø)=ASC(MID$(A$,2*M+1,1))+A
SC(MID$(A$,2*M+2,1))*256:NEXT
410 FOR M=1 TO 3:FOR N=Ø TO 3:READ A$:A(
N,M)=VAL("&H"+A$):NEXT:NEXT
420 DATA BASIC              :'TONE NAME
430 DATA ØØØØ,ØØØØ,ØØØØ,Ø:'TRPS,FB
440 DATA ØØØØ,ØØØØ,ØØØØ,Ø:'MODULATOR
450 DATA ØØ21,Ø1FØ,ØØØØ,Ø:'CARRIER
460 RETURN
```

確認用データ 使い方は71ページ

```
 10>Ce4Ø   20>d1RØ   30>bXl1   40>NtOØ
 50>1r51   60>M12Ø   70>LiNØ   80>HReØ
 90>HMØØ  100>vFVØ  110>pXXØ  120>ØejØ
130>gmX2  140>al8Ø  150>R5ØØ  160>PP3Ø
170>qM3Ø  180>VtfØ  190>l23Ø  200>YP9Ø
210>eK6Ø  220>in8Ø  230>Ø18Ø  240>6SmØ
250>ehLØ  260>jdAØ  270>OsWØ  280>rl2Ø
290>B2ØØ  300>7Tu2  310>FA3Ø  320>RE3Ø
330>fD8Ø  340>qjM1  350>CnsØ  360>lM9Ø
370>M2ØØ  380>jHBØ  390>7FØØ  400>bdS1
410>MgQØ  420>pN9Ø  430>1k8Ø  440>cKAØ
450>Uv8Ø  460>M2ØØ
```

⬆鳴り方とエンベロープの形を比べてみよう。画面の左から右までは2、3秒の時間的変化に相当するので、最初はキャリアを中心に注意深く視聴するとエンベロープがわかってくる

リスト5の使い方

【音色番号入力】実行すると、背景が灰色に変わり、「TONE NUMBER?」と聞いてくるので、音色番号を入力してリターンキーを押す。**【入力可能な番号】**ただし、「合成音色」と呼ばれる、1、7、8、11、13、15、17〜22、25〜32、34〜47、49〜63のみ受け付ける（つまり、音色一覧表で＊がついた15音は入力できない）。**【オリジナル音色を見る】**63を入力した場合は、行420〜450のデータによる音色になる。行420のデータ（8文字以内）が音色名になる。**【表示】**音色番号の入力が終わると、画面上部にモジュレータのエンベロープパターン、画面下部にキャリアのエンベロープパターン、その下に音色名を表示し、その音色を「O4A1」で鳴らす。演奏は行310のPLAY文による。**【リトライ】**スペースキーで最初から。このあと、ただリターンキーを押すと直前の音色になる。

画面の見方

●画面上部がモジュレータのエンベロープパターン、下部がキャリアのエンベロープパターン。●左から右へ時間の流れ、下から上へそれぞれのレベル（モジュレータは変調量、キャリアは音量）。●パターン途中にある赤い三角形はRR（リリースレイト）の始まりを表す。また、SL（サスティンレベル）の位置でもある。●画面最下行に表示される音色名は、それぞれ合成音色データ中にある正式なラベル。●画面右のオペレータ名の下にはそれぞれ図3の④〜⑥と⑦をそのまま表示している。

※リスト5では、表示上わかりやすいように実際の比率をわずかに歪めている。行30のPS＃の値が歪み率で、PS＃＝1のときに実際の比率になる。

たまにはプログラムでCGをかいてみたい

BASICピクニック

#27 伝統工芸としてのCG

音をたてずに走るように、相手に触れずに投げ飛ばすように、手を汚さずにお金をもうけるように、CGツールを使わずにCGをかいてみよう。

256色ぜんぶが使えるSCREEN8

今回は根気だけでCG[*1]をかいてみせよう。このページを見ているだけでは、どう「根気」が必要なのかはわからないだろうが、ページをめくるとどういうことかすぐにわかる。

そのまえに、その機能のわりに脚光をあびることがなぜか少ないSCREEN8を特別に紹介しておこう。余談だが、ピクニック担当者の3歳になろうとしている娘は「特別に」といおうとして「飛ぶケツに」という。

§

SCREEN8は、横256ドット×縦212ドットの画面構成で、解像度だけで見ればSCREEN5とおなじ。横が512ドットのSCREEN7に比べると、細かな表現には向いていないかに思える。少なくともグラフや線画では、やはりSCREEN7には勝てない。

しかし、SCREEN8は、256色(透明を含む)が同時に表示できる。えらそうにしているSCREEN7は、SCREENØ〜5とおなじで16色(透明を含む)しか表示できない。SCREEN7と8は、ドットの細かさでいくか、色数の多さでいくかの二者択一をせまられた結果、それぞれ片方をあきらめて進化した画面モードなのだ。

§

SCREENØ〜5、7は、16のパレット[*2]コードごとに赤、緑、青の輝度(各8段階)を調整して、512色中から好みの色あいを選び出すことができる。

SCREEN8には、そういうパレットのようなシチメンドウなものはなく、直接、256色[*3]のカラーコードで色を指定する仕組みになっている。

図1:SCREEN8におけるカラーコードの仕組み

1	Ø	Ø	1	1	Ø	1	Ø

=154
(ハムの色かなあ)

G(緑)の輝度 (上位3ビット)　R(赤)の輝度 (中位3ビット)　B(青)の輝度 (下位2ビット)

＊1 CG シージー。Computer Graphicsの頭文字をとって略したもの。「C.G.」ともかく。コンピュータを使ってかいたグラフィックのこと。そのまんまやないかい。広い意味では文字パターン定義したものや、文字をならべてなにかの形を表現したものも、CGといえなくもない。

＊2 パレット palette パレットの原義は、画家が使う例のアレ(絵具をまぜあわせる絵皿)。MSXでは、R(赤)、G(緑)、B(青)の輝度を8段階で調節して色を表現する(ただしMSX2以降の機種)。各パレットは0〜15のパレットコードで指定する。パレットコードといっても特別なものではなく、ふだんカラーコードといっているもののこと。パレットコードnの色あいは、
COLOR=(n, r, g, b)
を実行することで変更できる。ここでr、g、bは赤、緑、青の輝度でそれぞれ0〜7。8×8×8で512色を表現できることになる。

＊3 256色 512色でないのは、青の輝度が2ビット(4段階)しかないため。これは1バイト(8ビット)という限られた情報量のなかで3色ぶんの輝度情報を確保しなければならなかったため。ちなみに人間の目は青の輝度に鈍感だそうで、そのために青が節約をしいられたのだ。

＊4 光の3原色 赤(Red)、緑(Green)、青(Blue)のこと。RGBディスプレイのRGBはもちろんここから来た。ちなみに、光じゃないほうの色の3原色は、赤(マゼンダ)、青(シアン)、黄色(イエロー)。

リスト1：SCREEN8による256色のカラーチャート

```
100 SCREEN 8:COLOR 255,0,0:CLS
110 DEFINT A-Z:BX=256/16:BY=212/16
120 FOR I=0 TO 1
130 FOR J=0 TO 1
140   OX=BX*8*I:OY=BY*8*J
150   FOR G=0 TO 7
160   FOR R=0 TO 7
170     X=OX+BX*G:Y=OY+BY*R
180     C=G*32+R*4+I*2+J
190     LINE (X,Y)-STEP(BX-1,BY-1),C,BF
200    'DRAW"C0NM-=BX;,0 M+0,-=BY;"
210   NEXT
220   NEXT
230 NEXT
240 NEXT
250 GOTO 250
```

プログラム解説

●行100＝初期設定。白にあたるカラーコードが255なのに注目。また、SCREEN8にかぎらず、グラフィック画面では色設定のあとCLS（またはスクリーンモード設定）をしないと背景色や周辺色に変化がないので注意。●行110＝変数初期設定。BX、BYは、各色のタイルの横縦のドットサイズ。●行120〜130＝IとJのループ開始。Iはカラーコードのビット1、Jはビット0（青の輝度）に相当。また、大きく4ブロックにわかれる画面全体の構成のうち、縦方向管理がJのループ、横方向管理がIのループになる。●行140＝各ブロック単位の原点の座標OX、OYの設定。●行150〜160＝GとRのループ開始。それぞれ緑と赤の輝度を管理する。●行170＝各色タイルの開始座標X、Yの設定。●行180＝カラーコードC設定。●行190＝色タイルをかく。●行200＝「'」を消して、この行を復活させると、下の右側の写真のように各タイルが縁取りされる。GML（グラフィック・マクロ・ランゲージ。DRAW文のデータ）は各タイルの右と下のアウトラインを描く。●行210〜240＝NEXT。●行250＝表示のための無限ループ。

↑リスト1の実行画面。各色タイルの境界が錯覚のために盛りあがって見える

↑行200を復活させて実行した画面。色タイルが縁取りされて1つ1つの色が独立して見える

しかし、パレットがないといっても、SCREEN8のカラーコードは、光の3原色[*4]の輝度を調整するパレットとおなじような構造になっている。どういうわけか、緑(G)、赤(R)、青(B)の順で、そのうえ、青は2ビット（4段階）しかないが、それぞれの輝度を指定して色を決めるという点では、まったくおなじだ。

§

図1は、そのカラーコード（1バイト）を2進数8桁に分解して、SCREEN8におけるカラーコードの仕組みを目に見えるようにしたものだ。

上位3ビットが緑(G)の輝度、次の3ビットが赤(R)の輝度、残りの2ビットが青(B)の輝度を指定する。

つまり、たとえば、緑の色を強くしたいときは、32（&B100000）の倍数を強くしたい段階（8段階）だけ加えればいいし、赤の色なら4（&B100）の倍数を強くしたい段階だけ加えればいいのだ。

これを利用してSCREEN8のカラーチャート（一覧表）を表示するプログラムを作ってみた（リスト1）。

図2：カラーチャートの見方

画面は青の輝度によって大きく4つのブロックに分かれる。それぞれのブロックでは、左から右へ緑(G)の輝度が8段階で明るくなり、上から下へ赤(R)の輝度が8段階で明るくなる。カラーコードは次の式で計算できる。
緑の輝度（緑数字）×32＋赤の輝度（赤数字）×4＋ブロック番号（青数字）

＊1　ハンドデジタイズ　hand-digitize(？)　完全に手作業で点の座標データを取ること。もしかしたら、いま思いついた造語かもしれない。世間様では使わないように。

＊2　284×212　SCREEN 8は256×212ドットなのに、なぜ、方眼紙は、このドット構成でなければならないか。方眼紙上の絵を画面に移す場合、これは重要なポイントだ。

以前もBASICピクニックで触れたことがあるが、MSXの画面上のドットは、完全な正方形ではなく、横が縦の1.11倍大きい(編集部のディスプレイ上での実測)。そのために、たとえば、LINE文などで縦横おなじドット数の四角をかいても正方形には見えないし、円をかいても真円には見えない。横のドット数を1.11分の1(約0.9倍)するか、逆に縦のドット数を1.11倍するかしてやればドット比と実際のグラフィックとが一致する。

そこで、絵をハンドデジタイズする場合も、あらかじめX座標を1.11倍にした方眼紙上で書いておき、そうやって取ったデータをX方向のみ1.11分の1して画面上に再現していけばむだがない。もし、256×212の方眼紙を使うと、①絵が平たく見える、②X方向の座標を調整するために右側にアキができる、のどちらかになる。

＊3　ロジカルオペレーション付きCOPY文と裏ページ　ロジカルオペレーションは、転送するドットのカラーコードと転送先にあるドットのカラーコードとのあいだで論理演算をして、その結果を転送後のドットのカラーコードとするものだ。ここでは説明しきれないので自学自習してほしい。また、SCREEN 8はVRAMを大量に使うため、裏ページは1面しかない。色数が16色でもよければ、SCREEN 5で、裏ページ3面を使って、いろいろと遊びたいものだ。

＊4　リスト2　プログラムそのものは単純なのでポイントだけ解説する。

座標データを、たとえば「85－42」という形の文字にしてDATA文に置いてあるので、それを数値にもどすためにFNX、FNY(ユーザー定義関数)を定義している。そのときの変数A$から、ハイフンを目印にX成分、Y成分を切り出している。

行40～80は、白線で顔をかくデモ。座標のデータは1ドットずつ読みこんで描画している。データがひと区切りつくたびに「¥」でいったん線を切る。「¥¥」は、描画全体の終わりと色塗りの終わりを示す。

行90は、いったんかいた白線をカラーコード1で上書きして消す。

行110～150。先に眼鏡だけを白で塗っておき(行110)、眼鏡の周囲、すなわち座標(29, 91)から(153,157)までの長方形の区画を切り出すための座標変数を用意し(行120)、ページ1のおなじ場所に前ページのカラーチャートの4分の1をかき(行130)、白い眼鏡をANDでその模様の上にコピーし(眼鏡の部分だけ模様が残る)、それをもう一度ページ0のおなじ場所にTPSET(転送するグラフィックのカラーコード0以外の部分を切り抜いて上書き)でもどす(行140)。

あとは、てきとうなポイントを探してPAINTをくりかえし、できあがり。

「根気」でハンドデジタイズ

おじいさんやおばあさんにきくと教えてくれるかもしれないが、むかしは、パソコンのCGというと、下絵をかき、その絵をハンドデジタイズ[*1]して、座標データを取り出し、ＬＩＮＥ文やＰＡＩＮＴ文で絵をかくものと相場が決まっていた。

なんの因果か、わたしはそれを思い出し、いにしえの伝統工芸の技術の一端を誌面で再現してみたくなったというわけだ。

ほかのパソコンなみのグラフィック機能を持ったMSX2が、3大MSXメーカーのおかげで安価になり、一般に普及するようになったころには、すでに、ハンドデジタイズによるCGはすたれてしまっていたので、ＭＳＸでこの手のCGをやっていた人は少ないかもしれない。わたしだって、ハンドデジタイズCGを実際にやるのははじめてなので、あまり伝えられることは多くはないが、その伝統技法の輪郭くらいは伝わるだろう。

§

絵の題材はなんでもよかったので、そばでギョピギョピ騒いでいた学生アルバイトのOクンの似顔絵にした。Oクンは『きんぎょ注意報！』のギョピが好きなせいか、顔全体がギョピそっくりだ。そこで、一部、ギョピを参考にしながらかいた。

ハンドデジタイズCGのやり方は、箇条書きにすると、

①最低284×212[*2]マスある方眼紙に下絵をかく

②「あらゆる曲線は無数の線分によって構成されている」という信念に基づいて、てきとうな点の座標をはかっていく

③その座標データをDATA文として入力する

②と③に根気が必要だ。

座標データはX、Yの2元あるので、それをどう扱うかがちょっと問題だが、リスト2ではハイフンでむりやりつないで、1点の座標は1つの文字データという形にした。そして、この点の座標データを読みこみ、点と点をＬＩＮＥ文で結び、てきとうな囲われた場所にＰＡＩＮＴ文で色をぬるプログラムをかけばいい。あとは、プログラム上のくふうで、いろんなバリエーションが考えられる。

リスト2で、技法らしい部分は、眼鏡の模様に使った抜きあわせの手法だろう。

裏ページに模様(カラーチャート)をかいておき、白く塗った眼鏡をそのうえにANDでCOPYして、こんどはTPSETでもとの場所にもどす、という手順でやった。

このロジカルオペレーション付きCOPY文と裏ページ[*3]を活用すれば、自由な形の枠でできたアニメーションも可能だ。いろいろ試していいのができたら、CGコンテストに応募してみよう。そのときは、「CGコンテスト・伝統工芸係」と書きそえておけば、優遇してもらえます(たぶん)。

図3　伝統工芸CG『なぞのOクン』のための下絵

⬇ドット1つ1つの座標をはかり、メモし、描画の順番を決める。丸2日かかった

リスト2:『なぞのロクン』描画プログラムリスト

```
10 SCREEN 8:COLOR 255,0,0:CLS
20 KEY6,"COLOR 15,4,7"+CHR$(13)
30 DEFINT A-Z:DEFFNX=VAL(MID$(A$,1,INSTR
(A$,"-")-1))/1.11:DEFFNY=VAL(MID$(A$,INS
TR(A$,"-")+1))
40 '●●● DRAWING
50 RESTORE 190
60 READ A$:PSET (FNX,FNY)
70 READ A$:IF A$=>"¥" THEN IF A$="¥¥" TH
EN 90 ELSE 60
80 LINE -(FNX,FNY):GOTO 70
90 IF P=0 THEN COLOR 1:P=1:GOTO 40
100 '●●● PAINTING
110 A$="50-150":PAINT (FNX,FNY),255,1:A$
="100-100":PAINT (FNX,FNY),255,1
120 A$="29-91":X=FNX:Y=FNY:A$="153-157":
DX=FNX-X:DY=FNY-Y
130 SETPAGE ,1:FOR G=0 TO 7:FOR R=2 TO 7
:LINE (X+16*G,Y+13*(R-2))-STEP(15,12),32
*G+4*R+2,BF:NEXT:NEXT:SETPAGE ,0
140 COPY (X,Y)-STEP(DX,DY),0 TO (X,Y),1,
AND:COPY (X,Y)-STEP(DX,DY),1 TO (X,Y),0,
TPSET
150 RESTORE 370
160 READ A$:IF A$="¥¥" THEN 180
170 READ C:PAINT (FNX,FNY),C,1:GOTO 160
180 GOTO 180
190 '●●● DRAWING DATA
200 DATA 85-42,86-36,88-31,91-27,95-23,9
9-20,104-17,110-15,¥:'● 1 ● HAIR
210 DATA 85-42,77-35,71-32,65-31,59-30,5
6-30,50-32,45-35,¥:'● 2
220 DATA 85-42,87-39,91-34,96-29,103-26,
113-22,121-20,129-19,137-19,144-19,151-2
0,157-22,163-23,171-26,182-29,192-35,197
-39,202-44,202-44,206-48,209-53,212-60,2
16-72,216-90,214-110,213-118,210-132,208
-141,206-149,203-156,198-163,¥:'● 3
230 DATA 197-154,198-163,200-173,206-185
,200-179,193-170,196-177,202-187,194-180
,183-159,185-176,190-185,185-180,178-173
,175-166,171-157,168-148,166-138,¥:'● 4
240 DATA 178-173,176-171,170-166,168-164
,165-158,163-153,160-147,158-142,156-133
,155-133,154-129,152-123,153-117,153-108
,151-104,149-100,146-97,142-94,135-91,12
6-90,121-90,115-91,¥:'● 5
250 DATA 102-87,98-87,95-89,94-92,96-94,
101-94,115-91,132-84,136-81,135-78,131-7
6,102-87,98-83,95-81,91-78,88-75,86-72,8
2-67,81-62,82-57,¥:'● 6
260 DATA 81-62,82-69,80-75,74-80,67-83,6
1-88,56-93,69-97,71-99,70-101,50-100,36-
98,32-96,33-94,36-92,56-93,¥:'● 7
270 DATA 85-42,76-38,69-37,63-38,57-39,5
1-41,44-45,40-48,36-53,32-59,29-66,27-73
,26-80,26-86,27-92,28-98,30-104,32-108,3
4-112,¥,45-106,50-100,¥:'● 8
280 DATA 153-117,152-123,151-128,150-132
,147-136,144-140,141-143,137-145,131-147
,127-148,121-148,115-148,110-147,106-146
,102-145,97-141,94-138,91-134,89-129,88-
125,88-120,87-115,88-112,88-107,90-102,9
2-100,97-95,101-94,¥:'● 9 ● EYEGLASSES
290 DATA 89-129,88-134,86-143,84-146,81-
150,78-153,72-156,68-157,63-157,58-157,5
3-156,42-153,39-151,35-148,33-144,30-139
,29-128,29-124,29-120,31-115,34-112,45-1
06,56-104,67-103,77-104,83-107,86-111,87
-115,¥:'● 10
300 DATA 42-153,40-156,39-158,39-162,39-
165,41-170,42-172,45-176,49-179,54-181,5
9-182,63-183,80-183,85-185,90-186,95-187
,102-187,110-186,116-185,124-183,128-181
,131-179,139-179,145-178,153-175,159-172
,164-169,168-164,¥:'● 11 ● CHIN
310 DATA 216-90,219-86,222-83,227-82,232
-81,236-82,240-83,245-85,249-87,253-90,2
58-98,260-106,261-113,260-120,258-130,25
4-140,249-149,243-156,227-167,238-155,24
1-149,242-146,243-141,238-150,234-155,22
6-162,¥:'● 12 ● GYOPI'S TAIL
320 DATA 234-128,231-134,230-139,229-144
,226-162,224-157,223-152,222-145,222-137
,222-130,223-123,225-116,¥:'● 13
330 DATA 222-137,220-132,217-129,214-123
,213-118,¥:'● 14
340 DATA 55-122,58-121,63-121,66-123,69-
130,70-135,69-137,67-138,65-136,64-134,6
3-130,61-127,57-127,55-129,54-131,53-134
,52-137,51-138,48-138,47-134,49-128,52-1
24,55-122,¥:'● 15 ● EYES AND MOUTH
350 DATA 115-112,118-111,123-111,126-113
,129-120,130-125,129-127,127-128,125-126
,124-124,123-120,121-117,117-117,115-119
,114-121,113-124,112-127,111-128,108-128
,107-124,109-118,112-114,115-112,¥:'● 16
360 DATA 91-155,95-155,97-157,100-163,10
3-169,103-176,99-180,95-179,93-177,90-17
1,89-165,88-160,89-156,91-155,¥¥:'● 17
370 '●●● PAINTING DATA
380 DATA 57-125,36,117-113,36,95-170,29,
150-150,159,80-90,159,150-50,98,100-90,3
1,50-97,31,250-100,2,0-0,252,¥¥
```

確認用データ(使い方は67ページ)

```
 10>iE20    20>cJ60    30>K7t0    40>QU20
 50>kG00    60>Xp20    70>lAF0    80>gk20
 90>SP40   100>en20   110>2RX0   120>fSc0
130>JMU1   140>Gwf0   150>lB00   160>EY40
170>gU50   180>3G00   190>k040   200>Q1e0
210>SRT0   220>mTT6   230>rv23   240>vc34
250>K2T2   260>WWX1   270>mHc2   280>UG07
290>oUI5   300>TxS6   310>ajO6   320>JHU1
330>4IN0   340>64h4   350>jMf4   360>aEh1
370>EP40   380>faG1
```

↑リスト2のまえに作ったチェック用プログラムの画面。座標の微調整をする

↑リスト2の最終画面。実行すると最初白い線で輪郭をかき、いったん消してから色を塗る

目と耳で確かめながら音色作り

BASICテクニック

#28 FM音源音色エディタの使い方

ごくふつうのメロディーでも音色しだいで妙なる調べに変化していく。8月号掲載のプログラムをバージョンアップしてふたたび音色エディットの世界へ。

付録ディスク
→P116の「スーパー付録ディスクの使い方」参照

TONE EDITウインドウ
❶トランスポーズ　その音色だけの音の高さ。10進数で250が半音に相当
❷フィードバック　1桁の上位3ビット。弦楽器ふうの音色作りに有効
❸レベルキースケーリング／トータルレベル　レベルキースケーリング(KSL)は、音が高いほど音量が小さくなる楽器の性質をシミュレートするものだ。上位2ビットのみで、値が大きいほど傾向が強くなる／トータルレベル(TL)は下位6ビット。モジュレータのみに有効で、この数値が大きいほど最大出力が小さくなる
❹ビブラートetc.　この1桁の4ビットは下位から順に、KSR(音が高くなるほど立ち上がりが速くなる傾向)、EG(0＝減衰音、1＝持続音)、VIB(ビブラート)、TRM(トレモロ)の各スイッチ。たとえばEGを切り換えたいときは2をプラスマイナス、VIBを切り換えたいときは4をプラスマイナスすればいい
❺マルチプル　写真説明参照
❻～❾　下図参照

■図1 エンベロープパラメータ(ADSR)

※AR、DR、RRは設定値が大きいほど短い。0は変化なしを意味する

手ごたえを感じる音色作りシステム

FM音源の基本的な仕組みや構造については、8月号で頭が痛くなるほどやったので、今回は理論的な背景よりも、具体的に音色データのどこをどういじればどうなるのかを中心にやっていきたい。

そこで、8月号掲載の「MSX-MUSIC内蔵の合成音色を見学する」と称するプログラムに手を加え「音色エディタ」といってよいものに改造した(33ページのリスト「新・FM音源音色エディタ」。以下、エディタと略す)。

ただし、付録ディスクのTONEEDIT.BASというファイルは、8月号に掲載したプログラムとまったくおなじもので、このプログラムをロードし、33ページの青地の部分を追加・修正して、右ページの写真のような音色エディタができあがる。あらたに打ちこむ部分がけっこう多いが、飛ぶケツに許してください。

§

MSX-MUSICは、CALL VOICECOPYという命令を使って、あらかじめ用意しておいた配列のデータを音色番号@63に転送することで、いろいろな音を作り出せる。

エディタでは、2次元の整数型配列Aを使って、配列上でデータを修正し、それを@63にボイスコピーすることで音を作り出している。この配列上のデータ構造は32ページに掲載した。解説するスペースはないが、興味のある人はこのなかに立ち入ってみるとおもしろい。

この32バイトのうち、8バイトは音名だったり、未使用だったりするので、ほんとうの音色にかかわる部分は半分以下。エディタの「TONE　EDIT」というウインドウに表示されている16進数24個のデータ(12バイト)が純粋な音色データ部分だ。これでもまだ何か無意味な部分があるが、おおざっぱに計算すると、このデータの組み合わせは、約150,000,000,000,000,000,000,000(0が22個)とおりもある！

音色のエディットとは、この無限に近い組み合わせのなかから、美しい、またはおもしろい音をひろい出すことなのだ。

TONE NUMBER? 1

↑実行すると音色番号をきいてくる。たとえば、1(Piano 2)を入力

モジュレータ 0F30 10D9
キャリア 0010 F3B2
1=Piano 2 トランスポーズ=0C00 フィードバック= 8

↑その音色のデータとエンベロープの形を表示する

❶トランスポーズ
❷フィードバック
トランスポーズ、フィードバック
モジュレータ
キャリア
❸レベルキースケーリング トータルレベル
❹ビブラートetc.
❺マルチプル
❻SL
❼RR
❽AR
❾DR

TONE EDIT
0C00 0008 T.F
0F30 10D9 MOD
0010 F3B2 CAR
KTEM SRAD

→SHIFT+スペースキーで、エディット用のウインドウが出る。黄色いカギカッコがカーソル。スペースキー(またはSHIFT+スペースキー)で1桁ずつのデータを増減できる。それぞれのデータの意味はなかなかピンとこないだろうが、設定を変更して音の変化を聴く、という作業をなんどかやればコツものみこめてくる。とくに、⑤のマルチプル(基本周波数に対する倍率。ただし、0は0.5倍、Bは10倍、Dは12倍、Eは15倍を意味する)は、モジュレータとキャリアの組み合わせでいろいろ変化するので楽しい

TONE EDIT
0C00 0008 T.F
0C32 10D9 MOD
0011 F3B2 CAR
KTEM SRAD

↑データを変更して、その音を聴くには、ESCキーを押す

モジュレータ 0C72 31D1
キャリア 0011 F3B2
63=NEWTONE! トランスポーズ=0C00 フィードバック= 8

↑モジュレータのエンベロープが変わり音色も変わった。音色名の変化にも注目

■エディタの操作方法

最初に、

TONE　NUMBER?

ときいてくるので、1、7、8、11、13、15、17〜22、25〜32、34〜47、49〜63(マニュアルの音色番号表で「*」のついていない音色番号)のどれかを入力する。すると、その番号の音色を作るモジュレータとキャリアのエンベロープグラフとその他のデータを表示し(以下グラフ表示画面という)、その音色で「V15O4L1A」を演奏する(行310。ここのMMLを好きなメロディに変更しよう)。

63を入力したときは、行420〜450のデータで設定されるユーザー音色のエンベロープグラフを表示する。スペースキーを押すとふたたび音色番号入力にもどり、ここでただリターンキーを押すと、直前に選んでいた音色のグラフ表示画面にもどってふたたび音を出す。

ここまでは「TONEEDIT.BAS」単体で使える機能。次のエディットの操作は、右ページリストの追加・修正をしないと使えない。

■音のエディット

グラフ表示画面で、SHIFTキーを押しながらスペースキーを押すと、左上に「TONE EDIT」というウインドウ(純粋な音色データを表示)が現れる。カーソルキーで黄色いカギカッコ形のカーソルを動かし、変更したい数字にあわせてスペースキーを押すと、データが1ずつ増加し、SHIFTキーを押しながらスペースキーを押すと1ずつ減少していく。

データの変更が終わったら、ESCキーを押す。

すると変更したデータを、ユーザー音色番号63に「NEWTONE/」という音色名で転送し、その音を演奏し、エンベロープグラフをかいてグラフ表示画面にもどる。

グラフ表示画面のときにSELECTキーを押すと、右上の写真のようなメッセージが表示され、以下、写真の順のように進めていけばエディットした音のデータをBASICのDATA文の形で残しておける。

こうして音色ライブラリーを作っておくと楽しい。

■図2 配列A(N,M)のデータ内訳

第1添字(N)

第2添字(M)	0		1		2		3	
0	音色名2文字目	音色名1文字目	音色名4文字目	音色名3文字目	音色名6文字目	音色名5文字目	音色名8文字目	音色名7文字目
1	トランスポーズ		未使用	フィードバック	未使用		未使用	
2	KSL / トータルレベル	TRM / VIB / EG / KSR / マルチプル	サスティンレベル(SL) / リリースレイト(RR)	アタックレイト(AR) / ディケイレイト(DR)	未使用		未使用	
3	KSL / 未使用	TRM / VIB / EG / KSR / マルチプル	サスティンレベル(SL) / リリースレイト(RR)	アタックレイト(AR) / ディケイレイト(DR)	未使用		未使用	

```
EDITED TONE DATA
************************
START LINE NUMBER? ■
```

⬆SELECTキーでこのモードに。DATAを作りたい行の最初の行番号を入力

```
EDITED TONE DATA
************************
START LINE NUMBER? 1000
TONE NAME(WIDTHIN 8)? S-HARP
1000 DATA S-HARP
1010 DATA 0C00,0008,0000,0
1020 DATA 0C72,31D1,0030,0
1030 DATA 0011,F3B2,0070,0
Ok
■
```

⬆8字以内で音色名を入力。データが表示されたら各行でリターンキーを押すこと

```
LIST390
390 RESTORE 1000
Ok
LIST 1000-
1000 DATA S-HARP
1010 DATA 0C00,0008,0000,0
1020 DATA 0C72,31D1,0030,0
1030 DATA 0011,F3B2,0070,0
Ok
■
```

⬆これでデータがプログラムにリンクされた。行390の行番号を変えてまたRUN

```
モジュレータ
0C72
31D1

キャリア
0011
F3B2

63=S-HARP    トランスポーズ=0C00
             フィードバック=   8
```

⬆音色番号入力のときに63を入れると、自分でエディットした音色をその音名で呼び出すことができる

リストの確認用データ

使い方は67ページ

10>Ce40	20>d1R0	30>bXl1	40>NtO0
50>1r51	55>LU02	60>M120	70>LiN0
80>HRe0	90>HM00	100>vFV0	110>pXX0
120>0ej0	130>gmX2	140>al80	150>R500
160>PP30	170>qM30	180>Vtf0	190>l230
200>YP90	210>eK60	220>in80	230>0180
240>6Sm0	250>ehL0	260>jdA0	270>OsW0
280>rl20	290>B200	300>7Tu2	310>FA30
320>Zlp0	330>fD80	340>qjM1	350>Cns0
360>lM90	370>M200	380>jHB0	385>OX60
390>7F00	400>bdS1	410>MgQ0	420>pN90
430>1k80	440>cKA0	450>Uv80	460>M200
500>Bf10	510>uc80	520>nIF1	530>1xe0
540>7N30	550>7172	560>HXH0	570>NjD1
580>nv30	590>Itc2	600>lY70	610>Bi10
620>jXd1	700>nc30	710>l3J0	720>Te50
730>jaR0	740>eS60	750>Wwe1	760>9200

■リスト 新・FM音源音色エディタ 付録ディスク「TONEEDIT. BAS」＋追加・修正（青地の部分）

```
10 '●●● FIRST SETTING
20 CALL MUSIC:OPEN"GRP:"AS#1:DEFINT A-Z:
DIM A(3,3),D$(3,3),N$(1),NG(14),PR(4)
30 DEFFNP(N,M,L)=VAL("&H"+MID$(RIGHT$("0
00"+HEX$(A(N,M)),4),L,1)):PS#=1.5:DEFFNR
(I)=70*(2/PS#)^-PR(I):DEFUSR=342
40 FOR I=1 TO 14:READ NG(I):NEXT:DATA 2,
3,4,5,6,9,10,12,14,16,23,24,33,48
50 H=95:L=5:KO=10:KF=190:N$(0)="ﾓｼﾞｭﾚｰﾀ"
:N$(1)="ｷｬﾘｱ":LG#=LOG(10)/20/PS#
55 DIM EX(7),EY(2):FOR I=0 TO 7:EX(I)=6+
I*8-(I>3)*8:NEXT:FOR I=0 TO 2:EY(I)=14+8
*I:NEXT:DEFFNSHFT=PEEK(&HFBEB)AND1XOR1
60 '●●● INPUT
70 COLOR 15,14,0:SCREEN 0:WIDTH 40:KEYOF
F:A=USR(0):INPUT"TONE NUMBER";TN#
80 IF TN#>0 AND TN#<64 THEN TN=TN#:FOR I
=1 TO 14:IF TN<>NG(I) THEN NEXT:GOTO 100
90 BEEP:GOTO 70
100 IF TN=63 THEN GOSUB 380:CALL VOICECO
PY(A,@63) ELSE CALL VOICECOPY(@TN,A)
110 FOR M=0 TO 3:FOR N=0 TO 1:D$(N,M)=RI
GHT$("000"+HEX$(A(N,M)),4):NEXT:NEXT
120 TR$="ﾄﾗﾝｽﾎﾟｰｽﾞ="+D$(0,1):FB$="ﾌｨｰﾄﾞﾊ
ﾞｯｸ=    "+MID$(D$(1,1),4)
130 TN$=STR$(TN)+"=":FOR I=0 TO 3:CH$=RI
GHT$("000"+HEX$(A(I,0)),4):TN$=TN$+CHR$(
VAL("&H"+MID$(CH$,3,2)))+CHR$(VAL("&H"+M
ID$(CH$,1,2))):NEXT
140 '●●● BASIC LINES & TONE NAME
150 SCREEN 5
160 LINE(KO,0)-(KO,H*2),8
170 LINE(KF,0)-(KF,H*2),7
180 FOR I=0 TO 1:FOR J=0 TO 1:LINE(0,H*(
J+1-I)+L*I)-STEP(255,0),I+1:NEXT:NEXT
190 DRAW"BM12,193":PRINT#1,TN$
200 '●●● DRAWING & PLAYING(440Hz)
210 FOR OP=0 TO 1:FL=H-L:GOSUB 330
220 IF PR(3)=0 THEN 290 '● PR(3)=AR
230 LINE (KO,H+H*OP)-STEP(AR,-FL)
240 IF PR(4)=0 THEN LINE -STEP(KF-(KO+AR
),0):SL=0:RR=EG*RR:GOTO 260 '● PR(4)=DR
250 LINE -STEP(DR,SL):IF EG THEN LINE -S
TEP(KF-(KO+AR+DR),0)
260 DRAW"C9M+8,-6M-3,-2M-5,8" '● MARK
270 IF PR(2)=0 THEN LINE -STEP(255-(KO+A
R+DR),0):GOTO 290 '● PR(2)=RR
280 LINE -STEP(RR,FL-SL)
290 NEXT
300 COLOR 1,0:FOR OP=0 TO 1:PSET(196,11+
H*OP),0,TPSET:PRINT #1,N$(OP):FOR N=0 TO
 1:DRAW"BM+196,0":PRINT #1,D$(N,OP+2):NE
XT:NEXT:DRAW"BM116,193":PRINT #1,TR$:DRA
W"BM116,201":PRINT #1,FB$:COLOR 15,14
310 PLAY#2,"@=TN;V15O4L1A"
320 IF STRIG(0) THEN IF FNSHFT THEN 500
ELSE 70 ELSE IF ASC(INPUT$(1))=24 AND TN
=63 THEN 700 ELSE 320
330 '●●● PARAMETER READING SUB
340 M=OP+2:FOR I=1 TO 4:PR(I)=FNP(1,M,I)
:NEXT:AR=FNR(3):DR=FNR(4):RR=FNR(2):EG=S
GN(FNP(0,M,3) AND 2)
350 IF OP=0 THEN TL=(FNP(0,2,1)*16+FNP(0
,2,2)) AND 63:FL=FL*EXP(-TL*.75*LG#)
360 SL=FL*(1-EXP(-PR(1)*3*LG#))
370 RETURN
380 '●●● ORIGINAL TONE DATA SETTING
385 IF TN$=" 63=NEWTONE!" THEN RETURN
390 RESTORE 420
400 READ A$:A$=LEFT$(A$+SPACE$(8),8):FOR
 M=0 TO 3:A(M,0)=ASC(MID$(A$,2*M+1,1))+A
SC(MID$(A$,2*M+2,1))*256:NEXT
410 FOR M=1 TO 3:FOR N=0 TO 3:READ A$:A(
N,M)=VAL("&H"+A$):NEXT:NEXT
420 DATA BASIC           :'TONE NAME
430 DATA 0000,0000,0000,0:'TRPS,FB
440 DATA 0000,0000,0000,0:'MODULATOR
450 DATA 0021,01F0,0000,0:'CARRIER
460 RETURN
500 '●●● EDIT
510 SCREEN ,0:SPRITE$(0)=STRING$(7,1)+CH
R$(255)
520 LINE (3,3)-(113,51),12,BF:FOR I=0 TO
 3:PRESET(20+I,6-I¥2),0,TPSET:COLOR 1+14
*(I¥2),0:PRINT #1,"TONE EDIT":NEXT
530 FOR I=0 TO 3:PRESET(20+I,6-I¥2),0,TP
SET:COLOR 1+14*(I¥2),0:PRINT #1,"TONE ED
IT":NEXT
540 LINE (5,12)-(111,49),5,BF
550 FOR M=1 TO 3:FOR I=0 TO 1:PRESET(7+I
,7+M*8),0,TPSET:COLOR 15:FOR N=0 TO 1:PR
INT#1,D$(N,M)+" ";:NEXT:COLOR 9:PRINT #1
,MID$("T,FMODCAR",M*3-2,3):NEXT:NEXT
560 FOR I=0 TO 1:PRESET(7+I,40),0,TPSET:
PRINT#1,"KTEM SRAD":NEXT
570 S=STICK(0):EX=(EX-(S=3)+(S=7)+8)MOD8
:EY=(EY-(S=5)+(S=1)+3)MOD3:IF INKEY$=CHR
$(27) THEN 610
580 PUTSPRITE 0,(EX(EX),EY(EY)),10
590 IF STRIG(0) THEN N=EX¥4:M=EY+1:L=EX
MOD 4+1:UD=1-FNSHFT*2:MID$(D$(N,M),L,1)=
RIGHT$(HEX$(FNP(N,M,L)+UD),1):A(N,M)=VAL
("&H"+D$(N,M)):GOTO 540
600 TIME=0:FOR W=0 TO 5:W=TIME:NEXT:GOTO
 570
610 '●●● EXIT
620 COLOR 15,14:TN=63:TN#=TN:CALL VOICEC
OPY(A,@63):FOR I=0 TO 7:POKE VARPTR(A(0,
0))+I,ASC(MID$("NEWTONE!",I+1,1)):NEXT:G
OTO 120
700 '●●● DATA MAKING
710 SCREEN0:PRINT "EDITED TONE DATA":PRI
NT"**************************"
720 INPUT "START LINE NUMBER";LN
730 INPUT "TONE NAME(WIDTHIN 8)";TN$:TN$
=LEFT$(TN$+SPACE$(8),8)
740 PRINT MID$(STR$(LN),2);" DATA ";TN$
750 FOR M=1 TO 3:PRINT MID$(STR$(LN+10*M
),2);" DATA ";:FOR N=0 TO 2:PRINT RIGHT$
("000"+HEX$(A(N,M)),4);",";:NEXT:PRINT H
EX$(A(N,M)):NEXT
760 END
```

MSX製アナログ時計で世界の時を刻む

BASICピクニック

#29 アナログ時計の作り方

→P.118の「スーパー付録ディスクの使い方」参照

Mファンにディスクが付いたり、ソ連が解体したり、落合が三冠王を取ったり(予定)している、この激動の時を、MSX製アナログ時計で刻みたい。

タイムスタンプ　ディスクにセーブされているファイルは、更新されるたびにその更新されたときの日付と時刻の情報をディスクに記録している。これは、Disk BASICでは見ることができないが、MSX-DOSを立ち上げて、「DIR」(BASICのFILESに相当)というコマンドを実行すると、ファイルごとにその日付と時刻を表示してくれる。ずっとBASICばかりで使っていたとしても、ひそかに日付と時刻データを記録しているのだ。これで、似たようなファイルのどちらが新しいバージョンかとか、いつ更新されたかなどの確認ができる。しかし、内蔵時計が狂っていてはその利用価値は激減する。

TPSET　グラフィックのロジカルオペレーションのなかでもっともよく使われる。ある領域Aを別の領域Bにコピーする場合、領域Aの絵柄(透明色以外のカラーコードでかかれたもの)部分は領域Bの上にそのまま乗せられ、もともとBにあったものは消えてしまうが、領域Aで透明色(カラーコード0)だった部分は、領域Bにもとからあった絵柄がそのまま残る、というものだ。つまり、領域Aの絵柄だけを切り抜いて、領域Bに上塗りするということになる。

ちなみに、一見、2つの箱しか表示されなそうな次のプログラムを実行してみよう。無数の箱がどっと現れるはずだ。COPY文は奥が深い。

```
SCREEN5:LINE(1Ø,1Ø)-(5Ø,5Ø),,B:COPY(Ø,Ø)-(255,211) TO (3,3),,TPSET
```

アナログ時計にいたる3つの道

MSX2以降のMSXには、優秀なデジタル時計(日付機能付き)が入っている。

編集部の壁時計や多機能電話内蔵のデジタル時計なんかよりもよっぽど正確で、わたしたちの編集部でいちばん信頼できる。

DOSをよく使う人は、タイムスタンプを有効に使うためにこの内蔵時計のことを気にしている人が多いだろうが、BASICだけで使っているとそうでもない。たとえば、編集部の約10台のMSXは、わたしがCLOCK－ICのことを調べるための実験をするまで、どれもこれもばらばらの時刻を示していた。ようするに、内蔵時計をだれも使っていなかったのだ。

†

一方、わたしは夏休みからずっと、BASICピクニックで、ロジカルオペレーションを使ったグラフィック用COPY文の実用例をやろうと考えていて、そのテーマを探していた。とくに、切り抜き機能が便利なTPSETをやるつもりだった。できれば、アニメーションがいいのだが、と考えつつ、そうなると絵柄を作るのに根性がいるなあと悩んでいたのだった。

†

さらに一方、今月号の『体操人類マットマン』というプログラムで、以前、3Dグラフィック

ページØと1の切り換え　時計全体

10:08:56

↑31ページのリスト2の実行画面。文字盤と長針、短針はCOPY文、秒針はLINE文で表示している。サービスでデジタル時計もつけておいた。1秒ずつスムーズに時を刻んでいく

ページ切り換え　リスト２の行360では、ページ０とページ１のビジュアル、アクティブを毎回切り換えている。アクティブページとは、グラフィックがかきこまれるVRAMの領域、ビジュアルページとは、画面表示されるVRAMの領域のことで、この場合、時計の各パーツをコピーしている最中はまったく表示されず、そのまえにすでに完成していた時刻の時計画面だけを表示するようになっている。
もしページ切り換えをしなければどうなるか、行360を消去して動かせばよくわかるはずだが、豚時猫分ｎ秒から、豚時猫分ｎ＋１秒になるときも、秒針だけをかきかえればいいというものではなく、文字盤からすべてかきかえているのだ。かきかえる時間は、この時計の場合は１秒もかからないのだが、どうあがいても、人間の目にはチラついて見えるだろう。
このチラつきをなくすために、裏ページで時計を完成させてから、表にあった古い時刻の時計とさっと交換しているのだ。こうすると、じつにスムーズに時計の針が進んでいくように見える。
リスト２の改造点　この時計はまったく音がしないので、秒針の音を入れてみよう。たとえば、次の２行を追加すればいい。

```
115 A$="T200L32S0M300O5":PLAY A$,A$
535 PLAY"A","F+"
```

音はてきとうに変えてみよう。
また、たとえば、Ｃ＄にてきとうな時間を入れておいて、行370で時刻を読みこんだ直後に、

```
375 IF A$=C$ THEN 1000
```

を入れておく。
つまり、時刻Ｃ＄がやってきたら、行1000に書いてあった命令を実行するようにできるわけだ。
そのほか、動作中に時間を変更できるようにするとか、いろんな応用が考えられる。
また、時計の表示は１秒ずつずれるので時刻は１秒早くセットすること。

針だけをコピーするTPSET

ページ2に長針のグラフィック、ページ3に短針のグラフィック、配列B%に文字盤のグラフィックを用意したとする。
時間は、GETTIME A$とするたびに正確な時間が入手できるはずだが、そのまえに内蔵時計が世界の時間と一致している必要がある。ついでだから日付ごとあわせよう。
まえにもどこかでやったが、鮭年鯉月鯛日豚時猫分猿秒なら、
SETDATE"鮭／鯉／鯛"
SETTIME"豚：猫：猿"
でセットできる。漢字１文字は、数字2桁を意味し、DATEは「／」、TIMEは「：」で区切るという点が重要ポイント。
しかし、やっぱりついでに対話式（というほどでもないが）日付時刻設定プログラムを作っておいた。付録ディスクの「TOKEI.BPY」（おもにリスト2）の行1000からがじつは下のリスト１になっているので利用してほしい。使い方は、
RUN1000
を実行して、要求に応じてデータを入力していくだけ。

†

さて、時間のデータによって、それに応じたグラフィックの場所とコピー方向を決定し、
①まずB%、つまり文字盤をふつうの上塗りコピー
②時間に応じた短針パターンをA%に読みこんで、適切な方向でコピー。このときに、TPSETを付けるのがたいせつ（でないと醜い黒い四角が現れる）
③同様に長針をやはりA%を介してコピー
④最後に、秒に応じてLINE文で秒針をかく

†

大枠は以上のとおりだが、注意点が2つある。
１つは、①〜④をやっているあいだは見せないで、完成してから見せるということ。具体的にいうと、ページ０とページ１の2ページを使い、片方をアクティブ、片方をビジュアルに設定して、これを毎秒入れ換える。
もう１つは、秒針をかくときは、まさかいちいち座標を計算しているわけにはいかないので、秒針描画用の座標テーブル（配列SX、SY）を用意しておき、あらかじめ計算しておくということだ。

†

あとは左の傍注を参照して、てきとうに機能アップしよう。

↑文字盤のグラフィックの入った配列B%をふつうにコピー

↑A%に短針を１つ入れ、時間に応じた方向でTPSETのコピー

↑おなじように、A%に長針を１つ入れ、TPSETのコピー

↑最後に秒針をLINE文でかく。完成したところでビジュアルに切り換える。

リスト1のプログラム解説

●リスト１は、リスト2のオマケとしてファイル名「TOKEI.BPY」のなかに含まれている。以下は行番号ごとの解説。
1010　ユーザー定義関数の定義。FNA$は、FNB$を定義するためのもので、「ある文字列の右から2文字を取り出す（１文字しかなければ頭に０を付ける）」。FNB$は、あとで日付と時刻のデータを入れる配列A$を前提にして「添字X以降の配列A$を連続して3つ、文字X$でつなぐ」。ようするに、日付、時刻設定用の文字列を作る関数（行1130で使用）
1020　画面初期設定
1030〜1080　年、月、日、時、分、秒の入力を順に受け付ける
1090〜1120　確認
1130　日付、時刻設定（FNB$〜が、設定用文字列を作成）
1140　終了

リスト1　日付・時刻の設定　ファイル名：TOKEI.BPY

```
1000 '●●● DATE-TIME SETTING
1010 DEFFNA$(X$)=RIGHT$("00"+X$,2):DEFFN
B$(X,X$)=FNA$(A$(X))+X$+FNA$(A$(X+1))+X$
+FNA$(A$(X+2))
1020 COLOR 15,4,7:SCREEN 1:WIDTH 29
1030 INPUT "年";A$(0)
1040 INPUT "月";A$(1)
1050 INPUT "日";A$(2)
1060 INPUT "時";A$(3)
1070 INPUT "分";A$(4)
1080 INPUT "秒";A$(5)
1090 PRINT "これでいいですか。 よければ スペースキー、やり
なおすなら ESCキー"
1100 FOR I=0 TO 1
1110 I=-STRIG(0):IF INKEY$=CHR$(27) THEN
 1020
1120 NEXT
1130 SETDATE FNB$(0,"/"):SETTIME FNB$(3,
":")
1140 PRINT:PRINT "せっていしました。":END
```

リスト2 COPY文で作ったアナログ時計 ファイル名：TOKEI.BPY

```
10 '●●● FIRST SETTING
20 COLOR 15,0,0:SCREEN 5
30 OPEN "GRP:" AS #1
40 OX=9:OY=42:ZX=40:ZY=40:TX=128:TY=80:H
R=0:MN=0:SC=0:T=0:P=0:R=1.11:A$=""
50 DEFFNINS=INSTR(2,A$,"-")+INSTR(2,A$,"
+"):DEFFNX=VAL(MID$(A$,1,FNINS-1)):DEFFN
Y=VAL(MID$(A$,FNINS))
60 DIM HG(1,7,1),XZ(5),YZ(5),SN(15),CN(1
5),OX(15),OY(15),A%((ZX*ZY/2+5)/2),B%((2
*ZX*ZY+5)/2),HX(3),HY(3),BH(3),SX(59),SY
(59)
70 FOR I=0 TO 1:FOR J=0 TO 5:READ A$:HG(
I,J,0)=FNX:HG(I,J,1)=FNY:NEXT:NEXT
80 FOR I=0 TO 3:READ A$,A:HX(I)=TX+FNX/R
:HY(I)=TY+FNY:BH(I)=A:NEXT
90 TD=ATN(1)*2/15:FOR I=0 TO 15:SN(I)=SI
N(TD*I):CN(I)=COS(TD*I):NEXT
100 FOR I=0 TO 59:SX(I)=.8*ZX*SIN(I*TD)/
R:SY(I)=-.8*ZX*COS(I*TD):NEXT
110 FOR I=0 TO 15:OX(I)=OX+ZX*(I MOD 5):
OY(I)=OY+ZY*(I¥5):NEXT
120 '●●● HARI GRAPHIC PATTERN
130 FOR I=0 TO 1:SETPAGE I+2,I+2:CLS
140   FOR AN=0 TO 15
150     FOR J=0 TO 5
160       X=HG(I,J,0):Y=HG(I,J,1)
170       XZ(J)=X*CN(AN)-Y*SN(AN)+.5
180       YZ(J)=X*SN(AN)+Y*CN(AN)+.5
190     NEXT
200     FOR J=0 TO 5
210       X0=OX(AN)+XZ(J)/R:
          Y0=OY(AN)+YZ(J):
          X1=OX(AN)+XZ((J+1)MOD 6)/R:
          Y1=OY(AN)+YZ((J+1)MOD 6)
220       LINE (X0,Y0)-(X1,Y1),4+I*4
230     NEXT
240     PAINT (OX(AN),OY(AN)),4+I*4
250   NEXT
260 NEXT
270 '●●● BOARD
280 SETPAGE 1,1:CLS
290 CIRCLE (ZX,ZY),ZX-1,14,,,R:
    PAINT STEP(0,0),14
300 FOR I=0 TO 14:
      X=ZX+(ZX-5)*CN(I)/R+.5:
      Y=ZY-(ZY-5)*SN(I)+.5:
      IF I MOD 5 THEN PSET (X,Y),1
                 ELSE CIRCLE (X,Y),1,1,,
,R:PAINT STEP(0,0),1
310 NEXT
320 CIRCLE (ZX,5),4,1,,,2:PAINT STEP(0,0
),15,1
330 COPY (ZX,ZY)-STEP(ZX-1,1-ZY) TO A%:F
OR I=0 TO 3:COPY A%,I TO (ZX,ZY):NEXT
340 COPY (0,0)-STEP(2*ZX-1,2*ZY-1) TO B%
:CLS
350 '●●● CLOCK START
360 P=1-P:SETPAGE P,1-P
370 GETTIME A$
380 PRESET(100,20),,PSET:PRINT#1,A$
390 HR=VAL(MID$(A$,1,2)) MOD 12
400 MN=VAL(MID$(A$,4,2))
410 SC=VAL(MID$(A$,7,2))
420 '●●● BOARD
430 COPY B% TO (TX-ZX,TY-ZY),,PSET
440 '●●● HOUR
450 T=(HR+MN/60)*5:PH=T¥15:GOSUB 610
460 COPY (OX(AN)-7,OY(AN)+5)-STEP(ZX-1,1
-ZY),3 TO A%
470 COPY A%,BH(PH) TO (HX(PH),HY(PH)),,T
PSET
480 '●●● MINUTE
490 T=MN:PH=T¥15:GOSUB 610
500 COPY (OX(AN)-7,OY(AN)+5)-STEP(ZX-1,1
-ZY),2 TO A%
510 COPY A%,BH(PH) TO (HX(PH),HY(PH)),,T
PSET
520 '●●● SECOND
530 LINE (TX,TY)-STEP(SX(SC),SY(SC)),10:
    CIRCLE (TX,TY),2,15
540 GETTIMEB$:IF B$=A$ THEN 540
550 GOTO 360
560 '●●● HARI GRAPHIC DATA
570 DATA 0+2,-2+0,-2-29,0-31,2-29,2+0
580 DATA 0+2,-3+0,-3-19,0-21,3-19,3+0
590 '●●● COPY POSITION
600 DATA -7+5,2,-7-5,0,7-5,1,7+5,3
610 '●●● AN CALICULATING SUB
620 IF PH MOD 2 THEN AN=15-T MOD 15 ELSE
 AN=T MOD 15
630 RETURN
```

※行300は次の段に続

リスト2のプログラム解説

10〜110 初期設定
20〜30 画面設定／グラフィック画面に文字を表示する準備
40 変数設定
OX、OY＝針グラフィック描画時の基準座標（仮想的な原点）
ZX、ZY＝時計のサイズ（それぞれ中心点からの距離）
TX、TY＝時計を表示するときの中心点
HR、MN、SC＝時、分、秒
T＝針の角度（AN）計算用
P＝ビジュアルページ、アクティブページ切り換え用
R＝画面の縦横比の歪み修正用
A$＝時間データ読みこみ用
50 ユーザー定義関数定義
FNINS＝文字列A$のとちゅうにある正負符号の位置
FNX＝行570〜580、行600の座標データをX、Y座標用に切りわけるためのもの
60 配列宣言
HG(n, m, l)＝長針、短針の元グラフィックの座標データ。n：長針、短針／m：各頂点／l：X、Y座標の別
XZ(n)、YZ(n)＝針グラフィック描画時の仮想的原点からの増分。nは頂点番号
SN(n)、CN(n)＝6度ごとの三角関数値
OX(n)、OY(n)＝角度別針グラフィックの仮想的原点
A%(n)、B%(n)＝グラフィックコピー用配列
HX(n)、HY(n)＝4分割パート（0〜3時、4〜6時、7〜9時、10〜12時）ごとのコピー先座標⇒PH
BH(n)＝4分割パートごとのコピー方向⇒PH
SX(n)、SY(n)＝秒針表示用の座標増分。nは秒
70〜110各配列設定
TD＝30分の1π（6度）
120〜260 針グラフィックのセット作成（ページ2、3）
X、Y＝配列HGを一時代入
X0、Y0、X1、Y1＝個々の針グラフィック描画用座標。LINE文の始点と終点
270〜340 文字盤作成（ページ1）
280 ページ設定／画面消去
290 円盤をかく
300〜310 目盛りをかく
320 12時用目盛りをかく
330 目盛りをかいた文字盤の部分を配列A%を通して残りの4分の3にコピー
340 完成した文字盤全体を配列B%にコピーして画面消去
350〜550 時計の動き
360 ページ切り換え
370〜400 A$に時間を読みこみ、時、分、秒にわけて設定
410〜530 文字盤、短針、長針、デジタル表示、秒針、中央リングの表示
PH＝4分割パートの識別番号（右回りに0〜3）
540 時刻が更新されていなければ更新されるまで待つ（1秒に1回表示のテンポを守るため）
550 行360に行く
560〜580 長針、短針の元図のグラフィックデータ
590〜600 4分割パート別コピー先座標とコピー方向データ
610〜630 針の角度計算サブ

ターボRを持っているという幸せの科学

BASICテクニック

#30 ターボRのRAMディスクとは？

MSX FAN 付録ディスク
➔P116の「スーパー付録ディスクの使い方」参照

ターボRユーザー、または将来ターボRユーザーになるかもしれないあなたに。ターボRがターボRである存在理由の1つ、RAMディスクについて。

RAMディスク　RAMはRandom Access Memory(随時読み出し書きこみ記憶装置、などと訳されることもかつてはあった)の頭文字をとったもので、読み書きできるメモリ用LSIのこと。通常、ROM(Read Only Memory：読み出し専用メモリ)と対で使用される。そのRAMのなかにディスクドライブのような空間を作って、ディスクドライブとおなじような使い勝手でRAMを使えるようになっているものをRAMディスクという。欠点は、電源が消えると、それまで入っていたものがぜんぶ消えてしまうこと。ただし、最近流行しているノートパソコンでは、電源を切っても、ちゃんと記憶内容を保持しているものが多い。これをレジューム機能という。
A1STのRAMディスクは、標準で、最大96Kまで。GTは、最大352Kまで設定できる。

MSX－DOS2　日本語対応で、階層化ディレクトリが導入されているなど本格的なDOS(Disk Operation System)。ターボRには標準装備。

Kバイト　「キロバイト」と読む人もいるが、ふつうのキロと区別するために「ケーバイト」と読む人も多い。1Kバイトは1000バイトではなくて、1024バイトなのだ。ちなみに、1M(メガ)バイトとは、1024Kバイト。だから、A1STのRAM256Kバイトに、MEM－768のRAM768Kバイトを加えると、1024Kバイト、つまり1Mバイトになる。ところで、これを1エムバイトと読む人には出会ったことがない。

RAMのなかで起きている革命

RAMディスクがターボR独自のものであるかのようにいうのは、じつはちょっとまちがっている。

第1に、RAMディスクという機能は、MSX－DOS2(以下DOS2)に属するもので、したがって、MSX2でもDOS2用のカートリッジを増設していれば、RAMディスク機能が使用できる。ターボRはDOS2標準装備なので、RAMディスク機能も標準装備になっているというにすぎない。だが、やはり、RAMディスク機能を持つMSXは、圧倒的多数をターボRが占めている。

第2に、これとはまったく筋道がちがうが、ターボR、DOS2のRAMディスクのまえに「RAMディスク」と呼ばれるものがMSX2にあった。

この歴史的問題は、MSX2／2+ユーザーにとって、ターボRのRAMディスクを知るための大きな障害にさえなっていると思われるので、すこしくわしく書いておきたい(正直にいうとわたし自身がそうだった)。

■MSX2の「RAMディスク」

BASICで遊ぶMSX2以上ターボR未満のユーザーは、あるいはいまだにキツネにつままれたような気がしているかもしれない。いつのまにか、自分の機種にあったはずの「RAMディスク」が、DOS2やターボRの特別な機能のようにいわれているのだから。

残念だが、MSX2の「RAMディスク」は、現在、その名前で呼ばれていない。ターボR発表以降、名前は2度変わった。

ターボR発表時の本誌FFBの記事などを見ればわかるが、従来のMSX2などの「RAMディスク」は「メモリディスク」といいかえられ、ターボRではじめてRAMディスク機能が標準装備されたことになっている。

そのあと、ターボRのマニュアルを見ると、「メモリディスク」は「互換RAMディスク」に名前を変えられている。まえのシステム(MSX2／2+)で「RAMディスク」を使っていたプログラムと互換性がある、という意味だろう。

■名前が変わっていった理由

なぜ、こういうややこしいことになったのか。以下は、まったくの推測である。

そもそも、MSX2の「RAMディスク」は、その名で呼ぶにはふさわしくないものだった。

使ったことのある人はよくわかるだろうが、この「RAMディスク」は、ふつうのディスクとは使い方がかなりちがう。使うためにCALL MEMINI」を実行するなどはまだいいとしても、ファイル一覧表を見たり、

今回の記事に使用したプログラムは、リスト1～3がターボR専用、リスト5がA1GT(または新MSX-MUSIC)専用です。

■図1　記憶容量の比較

ユーザーエリア	「RAMディスク」(MSX2/2+)	RAMディスク(A1ST)	RAMディスク(A1GT)	2DDディスク
22〜24Kバイト	31.25Kバイト(32,000バイト)	96Kバイト(98,304バイト)	352Kバイト(360,448バイト)	713Kバイト(730,112バイト)

フロッピー>>>RAMディスク
ディスクをセットして
準備ができたらスペースキーを押してくだ
さい
RAMへの転送終了

RAMディスク>>>フロッピー
コピー用ディスクをセットして
準備ができたらスペースキーを押してくだ
さい

⬅リスト1の実行画面。最初に付録ディスクをセットしてスペースキーを押すと拡張子「FDZ」のファイル(13本)をぜんぶRAMディスクにコピーし、次にコピー先のフロッピーディスクを書きこみ可の状態にしたうえでセットしてスペースキーを押すと、RAMディスクからフロッピーディスクにコピーする。ディスクを何度も入れ換えずにすむので楽だ

ファイルを削除したりするには、それぞれ特別な命令を使う必要があった。さらに、ごくふつうにSAVEしても、なぜか、かならずアスキーセーブされ、COPY命令もランダムファイルも使えない。

そこへ、DOS2とともに、RAMディスクと呼ぶにふさわしい、ぜんぜん別のものが現れた。これが現在のRAMディスクで、あとで述べるが革命的に使いでがある。これをこそ、RAMディスクと呼ぶべきだ。そうなると、まえに「RAMディスク」と呼んでいたものに別の名前を用意しなければならない。そこで、「メモリディスク」という、よくわからない名前が出てきた。しかし、やはりその名前も、これまで長いあいだ「RAMディスク」と呼んできた歴史を振り返ると、すわりが悪い。そこで、「互換RAMディスク」という中間的な名前が最終的に考え出されたのではないか。

■これがRAMディスクだ

そういうわけで、ターボRのRAMディスクは、MSX2の元「RAMディスク」、現「互換RAMディスク」とは、まったくちがうものだということを基本的に認識しておきたい。

RAMディスクとは、使い方はディスクとおなじで、読み書きの速さはRAMとおなじ、というハイブリッドな存在なのだ。

このRAMディスクは、容量が小さく(図1参照)保存はきかないが、BASICでも幅広い利用法が考えられる。

たとえば、複数のファイルを一度にコピーするとき、1ドライブのユーザーは何回もディスクを抜き差ししなければならなかった。しかし、RAMディスクを使えば、たいてい1回ですむし、所要時間も短い。

今月の付録ディスクからファンダムのゲームプログラム(13本)だけを別のディスクにコピーする場合、リスト1を実行すれば左上の写真のような手順でさっさとコピーができてしまうのだ。これは、身近なRAMディスクの恩恵といえるだろう。

■リスト1　ある拡張子のファイルを一挙にコピーする　LIST1. BPZ

```
10 COLOR 15,4,7:SCREEN 0:KEYOFF:CALL KANJI1:WIDTH 38:LOCATE ,1
20 E$="FDZ" '拡張子の指定
30 COLOR 1,15:PRINT"フロッピー>>>RAMディスク":COLOR 15,4:GOSUB 120
40 CALL RAMDISK(0):CALL RAMDISK(96)
50 COPY "A:*."+E$ TO "H:"
60 PRINT"RAMへの転送終了"
70 PRINT:COLOR 1,15:PRINT"RAMディスク>>>フロッピー":COLOR 15,4
80 PRINT"コピー用";:GOSUB 120
90 COPY "H:" TO"A:"
100 SETBEEP3,4:BEEP:SETBEEP1,4
110 PRINT"全ファイルコピー終了":END
120 PRINT"ディスクをセットして":PRINT"準備ができたらスペースキーを押してください":FOR W=0 TO 1:W=-STRIG(0):NEXT:RETURN
```

リスト1の解説

●行10：画面初期設定。●行20＊：拡張子用変数E＄の設定。ここでE＄をたとえば「FMZ」にしておけば、FM音楽館の作品ファイルをぜんぶコピーするプログラムになる。●行30：見出しのメッセージ。●行40＊：ここがかんじん。「CALL RAMDISK(Ø)」はRAMディスク内の内容を消去する命令。次の命令で「96KバイトのRAMディスク」を設定する。96はA1STの最大値だが、GTはもっと大きく設定できる(図1参照)。●行50＊：付録ディスク(Aドライブ)からRAMディスク(Hドライブ)に拡張子E＄のファイルをすべてコピー。●行60〜80：メッセージ。●行90＊：RAMディスクからフロッピーディスクへコピー。●行100：音を出して注意。●行110：メッセージ。●行120：メッセージサブ。
※「＊」印の行以外はなくても可。

※リスト1は、付録ディスク収録の「LIST1．BPZ」と一部行番号が異なっています。

配列　配列にグラフィックデータを保持するためには、まず、配列宣言をしておかなくてはならない。
グラフィックに応じた必要な配列の大きさを計算する公式がある。横方向のドット数をX、縦方向のドット数をYとすると、SCREEN５の場合は、
DIM　A((X＊Y／C＋5)／D)
Cはスクリーンモードによって変わる定数で、５、７でC＝２、６でC＝４、８でC＝１。Dは配列の型によって決まる定数で、整約型はD＝２、単精度型はD＝４、倍精度型はD＝８。
これで配列AにCOPYできる。

リスト３についての補足

RAMディスクとBASICとCPUの相関関係　29ページのリスト３は、CPUを３つのモードに切り換えるものだが、CPUとはべつにBASICのバージョンによってRAMディスクが使えないので注意。いわゆるDOS１モード、つまり、Disk BASIC Ver.1.0で起動した場合は、このプログラムによってCPUをR800に切り換えてもRAMディスクは使えない。逆にDOS２モード、Disk BASIC Ver.2.01で起動しした場合は、CPUをZ80に切り換えてもRAMディスクが使える。

3つのモード　リスト３では、R800モードにモード１(ROM)とモード２(DRAM)の２種類あることになっている。これは、ROMターボ、RAMターボとも呼ばれるモードで、ふつうの「高速モード」はモード２を指す。ROMに入っているシステムをいったんRAMに移して、それで起動しているのがRAMターボ(モード２)で、ターボRではこの方式がいちばん速い。次に、RAMへの転送をおこなわずシステムをROMに置いたまま起動しているのがROMターボ(モード１)でやや遅い。RAMを使わないぶんRAMをほかの目的に使えるのだ。

モードランプ　ついでに行170の「128＋」を削除して引数をM％だけにすると、CPUが切り換わっても高速・標準モードを示すランプが切り換わらなくなる。

RAMのように高速に、ディスクのようにかんたんに

リスト２の実行画面。まず画面左上で各パーツを書いてRAMディスクにコピー

横に平べったい形からじょじょにふくらんで円になり、それをとおりすぎて縦に細長い形になり、そこからまたふくらんで……をスムーズにくりかえす

RAMであり、ディスクである、というRAMディスクの２面性は、COPY文を使ってみるとよくわかる。リスト１ではたんなるファイルのコピーだが、こんどはグラフィック用のCOPY命令を使う。

リスト２は、右ページの18枚の連続写真のように円が縦、横につぶれていく単純なアニメーションのプログラムで、すべての絵のデータをファイルとしてRAMディスクに保持し、そのファイルをおなじ座標に次々に呼び出してアニメーション効果を出している。

■配列とRAMディスク

COPYという命令は、
①画面(VRAM)にかかれているグラフィック
②メモリ(RAM)に配列で入っているグラフィックデータ
③ファイルとしてディスクに入っているグラフィックデータ
の３つともを扱うことができる貴重な存在なのだ。

配列のデータを使うということはメモリを呼び出すということなので、ほとんど時間はかからない。RAMディスクのファイルもそれとおなじようなものなので、配列と同様、呼び出しに時間がかからない。この点で配列とRAMディスクとはイメージが似ている。

しかし、配列を使ってリスト２とおなじ動きのプログラムを組むことはできないのだ。

なぜか。リスト２で扱っているグラフィック１つ１つはSCREEN５の61×61ドットのグラフィックで、これ１つぶんのデータ量は
61×61÷2＝1860.5
で1861バイト。一方、リスト２では32個もおなじ大きさのグラフィックを使っている。
1861×32＝59552
これは、58Kバイトを少し超える値だ。動いている最中のプログラムや変数、配列が置かれるユーザーエリアは余裕のある機種でも24Kバイトだから容量の点で配列は使えない。

容量の問題がなくても、グラフィックデータを配列にCOPYするには、そのデータの大きさに応じた配列宣言をする必要があるなど、配列は速いだけで使いにくい。

■フロッピーディスクとRAM

一方、形から見て、リスト２はわずかな修正だけで、RAMディスク用から通常のフロッピ

■リスト２　アニメーションのデータ基地にする　LIST2.BPZ

```
10 COLOR 15,4,7:SCREEN 5
20 CALL RAMDISK(0):CALL RAMDISK(96)
30 ST=1:GL=32:P=1:D$="H:"
35               'D$="A:"
40 FOR I=ST TO GL STEP P
50 CLS:CIRCLE (30,30),30,,,,I/(GL+1-I)
60 PAINT STEP(0,0),13,15
70 COPY (0,0)-(60,60) TO D$+MID$(STR$(I
),2)
80 NEXT
90 SCREEN 5
100 FOR I=ST TO GL STEP P
110 COPY D$+MID$(STR$(I),2) TO (80,80),
,PSET
120 NEXT:SWAP ST,GL:P=-P:GOTO 100
```

リスト２の解説

●行10：画面初期設定。●行20：RAMディスク初期化。最初は以前あったRAMディスクの内容を全削除するものなので注意。次は標準装備のA1STのRAMディスク最大値(96K)のRAMディスクを設定する命令。●行30：変数初期設定。ST＝ループの開始値、GL＝ループの終了値、P＝ループのステップ数、D$＝ドライブ指定用(ここではRAMディスク)。●行35：ドライブ変更用。●行40〜80：少しずつ歪み率のちがって円をかき、色を塗り、それをファイル名「1」〜「32」でディスクにコピー。●行90：中断したあとはプログラムを書きかえるなどしていなければ、「GOTO90」で再開する。●行100〜120：アニメーション。

↑リスト２実行中の連続写真。左上すみから右へ進む。全32カットのアニメーション。ターボＲの高速モードでやって32カットを往復するのに7.45秒、リスト３でＣＰＵをＺ80に切り換えて(ただしＢＡＳＩＣはVer.2.0Iでなくてはいけない)やってみると37.4秒

ーディスク用に変更できる。

行35の「'」(ＲＥＭ)を削除すれば、D$="A:"が生き、行70、110とも、Ａドライブを相手にするようになる。

じっさいにそのように修正し、Ａドライブに書きこみ可にしたディスク(付録ディスクは不可)を入れて実行してみよう(行20を削除すれば、この修正でMSX2／2+でも動く)。

アニメーションがきわめて遅くなり、ガリガリとディスクを読む音がしてうるさいだろう。

フロッピーディスクは読み書きが遅く、うるさい。一方、ＲＡＭディスクは、速く静かだ。

ちなみに、ＲＡＭディスクが速いのはR800だけのせいではない。試みに、リスト３によって高速モードから標準モードに切り換えて、プログラムのロード、セーブをやってみよう。やっぱり速い。

■16ビット化以上の意味

そういうわけで、ターボＲのＲＡＭディスクは、ユーザーエリアを使うときの面倒もなく、フロッピーディスクの遅さにもわずらわされない理想的な記憶スペースなのだ。

今回の記事は、Ａ１ＳＴを基準に作ったが、新機種のＧＴ(ＲＡＭ512K)では、27ページの図１のようにフロッピーディスクの約半分の容量を持つＲＡＭディスクが使える。これは、一見地味な内容だが、８ビットが16ビットになった以上の変化をもたらすような気がする。

きわめてついでに、ＧＴの新ＭＳＸ－ＭＵＳＩＣの一端を紹介しておこう。右のリスト４は、78ページに掲載されているＦＭ音源用プログラムだが、Ａ$とＢ$の頭のほうを修正し、#1を#2に変えると、それだけでＭＩＤＩ音源(この場合はＭＴ－32で試した)を鳴らすプログラムになる。これにも深いものがありそうなのだが、機会があればいずれまた。

↑ＲＡＭディスクのドライブ名は「Ｈ」。リスト２を実行したあとに、Ｈドライブのファイル一覧を見ると、グラフィックのファイルが32個できている。リスト２の行110は、このファイルを１つ１つ呼び出しているのだ

■リスト３　R800とZ80を切り換える　CPU-CHGR. BAS

```
10 'ターボ゛R センヨウ     モート゛ キリカエ
20 'ターボ゛R ノ CPU ヲ キリカエル
30 HI=PEEK(&HFC4A)+PEEK(&HFC4B)*256
40 CLEAR 200,HI+10
50 HI=PEEK(&HFC4A)+PEEK(&HFC4B)*256
60 RESTORE 180
70 FOR I=0 TO 6
80       READ A$
90       POKE HI+I,VAL("&H"+A$)
100 NEXT
110 DEFUSR=HI
120 COLOR 15,4,7:SCREEN 1:WIDTH 29
130 PRINT "0=ヒョウシ゛ュン モート゛(CPU=Z80A)
140 PRINT "1=コウソク モート゛1(ROM) (CPU=R800)
150 PRINT "2=コウソク モート゛2(DRAM)(CPU=R800)
160 INPUT "モート゛ ヲ エランデ゛クダ゛サイ";M%
170 A=USR(128+M%)
180 DATA 23,23,7E,CD,80,01,C9
```

リスト３の解説

●行10～20：ＲＥＭ文。●行30：現在のユーザーエリアの上限値をＨＩに設定。●行40：10バイトのマシン語領域を確保。●行50：またユーザーエリアの上限値をＨＩに設定。●行60～行110：マシン語書きこみ(ターボＲのＣＰＵを切り換える)、設定。●行120：画面初期設定。●行130～160：メッセージ表示、入力受け付け。●行170：マシン語実行。●行180：マシン語データ

■リスト４　FM音源用のプログラム　MOTHER. FMZ

```
10 CLEAR 500:CALL MUSIC(0,0,1,1)
20 A$="@3605L4T120V15A>CE2G8E8EE8D8D8C8C
8<A.A>CDE12D12C12EE8C8C8<A8AA2"
30 B$="@3605L4T120V15AA>C2<EGFEAA8G8FF8E
8DECFGAA2"
40 PLAY#2,A$
50 PLAY#2,A$,B$
60 GOTO 40
```

■リスト５　MIDI用のプログラム例(MT-32用)

```
10 CLEAR 500:CALL MUSIC(0,0,1,1)
20 A$="@H2@3205L4T120V15A>CE2G8E8EE8D8D8
C8C8<A.A>CDE12D12C12EE8C8C8<A8AA2"
30 B$="@H3@3205L4T120V15AA>C2<EGFEAA8G8F
F8E8DECFGAA2"
40 PLAY#1,A$
50 PLAY#1,A$,B$
60 GOTO 40
```

「ファイル」を使ってなにができるか

BASICピクニック

#31 シーケンシャルファイル、ランダムファイル

オープン。そういえば「ファイル」という妖怪がいた。ディスクのなかとはかぎらずいろんなところに現れ、そのうえ、2種類いるらしい。クローズ。

あやしい「ファイル」

■ファイルの意味をめぐって

ファイルということばは、いろんなところで使われ、いろんな意味を持っている。もとは1つの意味からわかれ、いいように使われているうちに妖怪のようなことばになってしまったのだが、もっとも重要でかつおもしろい「ファイル」だけにかぎっても、けっこう妖怪である。

今回のピクニックは、この妖怪の正体をみきわめるための旅になる(予定)。みきわめるだけなので、細かい話はどんどんすっとばし、1つ、2つのサンプルにとどめた。

今回のBASICピクニックで使用しているプログラムは、すべてディスクに収録されていない。これにはいろんな理由があるが、ようするにまにあわなかったのである。この付録ディスク時代にまったく申しわけないがそれぞれ打ちこんでほしい。

たんに、ファイルというと、わたしには、おもに「整理された書類の束」というイメージがあるのだが、どうだろうか。

英和辞典とか国語辞典にはどう書いてあるか。

●小学館ランダムハウス英和大辞典／第1版(小学館)

file [fail] ***n.,v.***(files,fil・ing)—— ***n.*** 1(書類・手紙などを整理して入れておく)整理保存用具、ファイル(紙ばさみ・状差し・書類とじ・小箱・整理だんすなど). 2(整理された新聞・書類・記録などの)とじ込み……(中略) 7 目録、名簿(list,roll)

●広辞苑／第1版(岩波書店)

ファイル【file】書類綴(しょるいとじ)。書類挟(ばさみ)。状さし。綴込(とじこみ)書類。(ふりがなは編集部)。

もともと、ファイルとは「紙ばさみ」のような保存用具のことをいい、それが紙ばさみで綴じてある「書類」のほうへも意味が広がったのだろう。こういうふうに、あるものの一部分の名称(ファイル=紙ばさみ)を使ってそれ全体(紙ばさみでとじられた書類)を呼ぶことを換喩(かんゆ)という。一般に、用語の混乱は換喩のせいであることが多い。

■ファイル=整理だんす

このようにもともと混乱気味の用語をそのままコンピュータに持ちこんだのが、これから探りにいく「ファイル」だ。日常生活では多少混乱した状態で使われても実害は少ないが、コンピュータの世界で混乱していてはちょっと困る。

コンピュータのなかでも日常的に使うファイルと、プログラムのなかで使うファイルというふうに、大きくわかれている。問題は後者で、適当な名前を考えてみると、先に見たランダムハウスのなかにあった「整理だ

ファイル　わたしがたまたま見たマニュアルには「ファイルとは意味を持つ情報の集まりのことです」と書いてあった。これは半分まちがっている。たしかにその意味で使うことも多いが、プログラム中では、「ファイルとは意味を持つ情報の集まりを入れる入れ物のことです」、が正しい。

互換RAMディスク　MSX2からある旧RAMディスク。この互換RAMディスクは、使用するファイル名のまえに「MEM:」というデバイス名を入れて、あるていどフロッピーディスクのように使える。ただし、プログラムファイルの場合は、アスキーセーブしかできない(ふつうにセーブしてもかならずアスキーセーブになる)し、データファイルの場合はシーケンシャルファイルだけが使える。

シーケンシャルファイル　シーケンシャル(sequential)には「順番に」という意味がある。文字どおり、つねに順番にデータを取り扱うファイルだ。

ランダムファイル　ランダムアクセスファイルともいい、そのファイルのどこにデータがあっても、みなおなじようにアクセスできるファイルのことをいう。このランダムは、RAM(ランダムアクセスメモリ)の「ランダム」とおなじ気持ちで使われている。

■リスト1　シーケンシャルなグルグル

```
10 DEFINT A-Z:SCREEN 0:COLOR 15,4,7:PRIN
T "SELECT >>> SETTING":PRINT "ANY    >>>
 GURUGURU":IF INPUT$(1)=CHR$(24) THEN 60
20 SCREEN 5:OPEN"MEM:DATA" FOR INPUT AS
#1:PSET (133,100)
30 INPUT #1,X,Y:LINE -(X,Y)
40 IF EOF(1) THEN CLOSE ELSE 30
50 IF STRIG(0) THEN 20 ELSE 50
60 CALL MEMINI:OPEN "MEM:DATA" FOR OUTPU
T AS #1:P#=ATN(1)*4/18:R#=5
70 FOR I=0 TO 216:R#=R#+.3:X=128+COS(I*P
#)*R#:Y=100+SIN(I*P#)*R#:PRINT #1,X,Y:NE
XT:CLOSE:SETBEEP4:BEEP:SETBEEP1:GOTO 10
```

リスト1の解説

【使い方】SELECTキーを押すとデータの計算とファイルへの保存がはじまり、終わるとピンポンと鳴る。最初の画面にもどるので、あとは、スペースキーを押すたびに渦巻きをかく。リセットか互換RAMディスクをいじるかしないかぎり、ずっとデータは生きている。

【プログラム解説】●行10＝初期設定ほか／入力待ち(SELECTキーなら行60へ。それ以外は次の行へ)●行20＝画面設定／互換RAMディスクの「DATA」を、入力用に#1として開く／初期座標指定●行30＝ファイル#1から、X、Yのデータを読みこみ、座標を線でつなぐ●行40＝ファイルが終わりなら閉めて次の行へ、まだデータが残っていれば行30へ●行50＝スペースキー入力待ち(押されれば行20へ)●行60＝互換RAMディスク初期化／互換RAMディスクのなかに「DATA」という名前で出力用に#1として開く／変数初期化(P#＝10度をラジアンに変換した数値、R#＝渦巻き描画用の半径初期値)●行70＝渦巻き用データ計算／ファイルを閉める／終了音／行10へもどる

んす」である。

プログラムにおけるファイルとは、データの集まりが主体ではなく、その入れ物や形式のほうに意味の重点がある。入れ物のことをいっているのか、データの集まりのことをいっているのかピンと来なかったら、迷わず入れ物と思えばまちがいない。

さて、ようやく具体的な話に入ることにしよう。コンピュータで使うファイルは、

①プログラムファイル
②データファイル

の2種類がある。プログラムファイルとは、BASICやマシン語のプログラムを保存するもので、データファイルとは、プログラムで使うデータを入れたり出したりするものだ。

しかし、たんにファイルというときは、①と②をあわせていっているか、②だけの意味で使っているかのどちらかだ。とくに、プログラムのなかで「ファイルをオープンして……」などというときは、100パーセント、②の意味だけで使っている。

わたしがこの記事で探ろうとしていた「ファイル」も、じつはこの「データファイル」だったのである。

このデータファイルも、データを受け入れることしかできないファイルや、引き出しが1つしかないファイル、たくさんあるファイルなど、いろいろある。

ファイルを置く場所や具体的機能などを問わず、全体の構造で分類してみると、データファイルには、

①シーケンシャルファイル
②ランダムファイルの2とおり

がある。それぞれどういうものかは、図1の解説と次ページで説明する。

リスト1は、あらかじめ渦巻きをかくための座標217組を計算して互換RAMディスク内に設けたシーケンシャルファイルに保存しておき、あとでそのファイルから取り出しながら一気にかくプログラムだ。データを保存したり、取り出したりするという点で、ファイルの典型的な使用例といえるだろう。

⊕先にデータを計算してシーケンシャルファイルに入れ、それを取り出しながら描画。速い

■図1　シーケンシャルファイルとランダムファイルのデータ構造例

●シーケンシャルファイル

レコード1	レコード2	…										EOF

●ランダムファイル

	変数1	変数2	変数3
レコード1			
レコード2			
レコード3			
：	：	：	：

■シーケンシャルファイル　たとえていえば、たった1個の引き出ししかない整理だんす。引き出しのなかに、さまざまな仕切りがあり、仕切り1つ1つが「レコード」と呼ばれる。データを入れるときは、手前から奥へ順番にしか入れられず、取り出すときもやはり手前から奥へ順番にしか取り出せない。

■ランダムファイル(ランダムアクセスファイルともいう)　引き出しをいくつも持った整理だんすで、引き出し1つがレコードに対応する。引き出しのなかは専用の文字変数に対応するように仕切られていて、その仕切り方も大きさもすべての引き出しがおなじ規格になっている。ある引き出しにデータを入れた直後、離れたべつの引き出しからデータを取り出すなど、データの入出力が自由自在にできる。

ファイル番号　「～AS　#1」の「#1」がファイル番号。シーケンシャルファイルにせよ、ランダムファイルにせよ、いちどファイルを開いたあとはこのファイル番号で指定して読み書きする。ファイル番号は、「オープン可能なファイルの数」までしかつけられない。「オープン可能なファイルの数」とは、そのときいちどに開けられるファイルの数ということで、電源オンしたときは1。これを増やすには、
MAXFILES＝n
という命令（nがファイルの数）で設定すればいい。ファイルを活用するようになると、どうしても複数ファイルをいちどにあけておく必要が出てくるため、この命令を避けて通れない。

データファイル　ここではおもにディスクのなかに作られるファイルのことばかりを書いているが、データファイルのなかには、出力専用のファイルもある。その1つが、よく使われる
OPEN"GRP：" AS #1
というもので、これはグラフィック画面（デバイス名「GRP：」）をファイルとしてオープンしている。ファイルスペックが省略され、ふつうなら必要なモード（FOR OUTPUT）も省略されているが、これもシーケンシャルファイルの1種なのだ。
このファイルにたとえば、
PRINT #1，"ABC"
などと出力すれば、グラフィック画面上に「ABC」という文字が現れる。
また、同様にテキスト画面「CRT：」やプリンタ「LPT：」なども使える。

バッファ　ランダムファイルは、同一規格のレコードを複数個、それぞれ並列して扱っている。レコードにデータを書きこむ命令がPUT、レコードからデータを取り出す命令がGETだが、このとき、どちらの場合もいったん、メモリのある部分を中継地点として使用する。この中継地点をバッファという。ランダムファイルでは、FIELD文で、そのバッファにいくつかの文字変数を割り当て、以後その文字変数が、ファイルとプログラムとのあいだにたって、データのやりとりをしてくれる。以上、図2参照。

ランダムファイルの日記帳

■OPENされるファイル

今回の記事の中心テーマになっている「ファイル」、つまり「整理だんす」と訳したほうがわかりやすいと思われる「ファイル」は、OPENという命令に関わるものだ。

ファイルとしての整理だんすには、ふだんシャッターが降りている。そこで、ファイルを使うには、シャッターを開けるところからはじめなくてはいけない。シャッターを開ける命令が
OPEN～
という命令だ。この命令のあとに続く「～」には、ファイルスペック（ファイル名と拡張子をあわせたもの）に続いて、ファイル形式やファイル番号（ファイルの整理番号）の指定が来る。ここで、開くファイルの性質が決められるのだ。

そのファイルの性質に2種類あり、それが前ページで述べたシーケンシャルファイルとランダムファイルであった。

つまり、シーケンシャルファイルか、ランダムファイルかは、OPENしたときにはじめて決まるものであり、ディスクに入っているデータがどういうものかとは関係ない。

さきほどの比喩を続けていうなら、シャッターを開けるときにとなえた呪文に応じて、現れた整理だんすが、たとえ、おなじファイルスペックでも、奥の深い引き出しが1つしかないものだったり、おなじ規格の引き出しがいくつもあるものだったりするというわけだ。

ここからしても、データファイルに関するかぎり、ファイルとはデータの集まりではなく、扱い方の形式（整理だんすの形状）のことなのだということがわかるだろう。

■2つのファイルの長所・短所

シーケンシャルファイルとランダムファイルは、どういう違いがあり、どういうふうに使いわけるのか。

そのまえにどうやって使うのかを説明していないが、これは、それぞれのファイル形式ごとにくわしくやるほかないので、使い方の説明は、次回にまわすことにしたい。

さて、どう違うのか。

シーケンシャルファイルは、使い方はかんたんだが、単純にしか使えない。ランダムファイルは、使い方はやや複雑だが、柔軟で幅広い使い方ができる。

たとえば、シーケンシャルファイルには、数値も文字も入れられるし、入れ方も表示のときに使うPRINT文とほとんどおなじ命令で、わかりやすい。取り出すときも、ただ命令をくりかえすだけで次々にデータを取り出せる。

使い勝手は、READ、DATA文にとてもよく似ている。違うのは、DATA文はプログラム自身が書き換えることはできないが、シーケンシャルファイルはデータを追加したり、新しい内容にしたりすることがかんたんにできるという点だ。

しかし、そうはいっても、頭から順にしか取り出せないし、ファイルにデータを入れるときと、ファイルからデータを取り出すときとは、OPEN文で指定しなおさなくてはいけない。

一方、ランダムファイルは、自由にデータを出したり入れたりできる。いったん開いてしまえば、やや使い方は限定されているものの、メモリのように自由な読み書きができる。そんなランダムファイルの利用例がリスト2、3の200字日記だ。これは、以前、プロコレで掲載したものを改造したものだ。

ただし、ランダムファイルにも、文字型しか扱えない（数値を扱うときは特別な関数を使う）とか、あらじめ決めておいた文字型変数を通じてしか、ファイルへの読み書きができない、フロッピーディスクかDOS2のRAMディスク（互換RAMディスクは不可）でしか使えないなどの制約がある。

などと、抽象的な話ばかりで終わってしまったが、次回は、まず、シーケンシャルファイルに的をしぼって、その楽しい使い途（みち）を考えてみよう。

■図2　ランダムファイルのデータのやりとり

■リスト2　日記帳をディスクに作る

```
10 '●●● ニッキ ノ ショキカ
20 CLEAR 800:SCREEN 1:WIDTH 29
30 PRINT "にっきを しょきかします"
40 PRINT "よういは いいですか(Y/N)":A$=INPUT$(1)
50 IF NOT(A$="Y" OR A$="y") THEN PRINT "
やめました":END
60 PRINT "にっき しょきか中 30秒ていど まってください"
70 OPEN "DIARY.TXT" AS#1 LEN=200:FIELD #
1,200 AS A$:FOR I=1 TO 12*31:LSET A$=STR
ING$(200," "):PUT #1,I:NEXT:PRINT "しゅうりょ
う":CLOSE
```

リスト2の解説

【使い方】リスト3で読み書きするための日記帳をディスクのなかに作るためのもの。ディスクをAドライブに入れ、

PRINT　DSKF(1)

を実行して出てきた数値が、73以上なければそのディスクには日記帳を作ることができないので注意。できるだけ、空き容量の多いディスクを使ってほしい。さて、リスト2の使い方は、実行したあとメッセージに従って適切なキーを押す。最後のところでは、Y(ソ)キー以外を押すと、ぜんぶNキー(ノー)を押したことになってしまうので注意してほしい。

【プログラム解説】●行10〜50 日記の初期化準備／最後にY(ソ)キー以外のキーが押されたらやめてしまう●行60　メッセージ●行70「DIARY.TXT」というファイルスペックで1レコード200バイトのランダムファイルを開く／1年ぶん(じっさいにはそれ以上)のディスクの記憶領域にスペースを埋めて領域を確保／ファイルを閉じて終了。

■リスト3　日記の読み書き

```
10 '●●● ショキセッテイ
20 CLEAR 800:COLOR 15,4,7:SCREEN 1:KEYOF
F:WIDTH 20:ON STOP GOSUB 140:STOP ON:DEF
FNK=PEEK(&HFAFC)
30 '●●● ランダムファイル ノ セッテイ
40 OPEN "DIARY.TXT" AS #1 LEN=200
50 FIELD #1,200 AS A$
60 '●●● ニッキ ノ ヨミカキ
70 CLS:INPUT"ひづけ(月,日)";M,D:A=M*31+D-31
80 IF M<1 + M>12 + D<1 + D>31 THEN 70
90 POKE &HFAFC,FNK AND 254:INPUT "よむ(R)¦
かく(W)";R$:ON INSTR(" RrWw",R$)¥2+1 GOTO
90,100,120
100 '●●●●● ヨム
110 GET #1,A:CLS:PRINT USING "##月##日";M;
D:PRINT A$:LOCATE 5,20:PRINT "HIT SPACE"
:FOR I=0 TO 1:I=-STRIG(0):NEXT:FOR I=0 T
O 1:I=-(INKEY$>""):NEXT:GOTO 60
120 '●●●●● カク
130 CLS:PRINT USING "##月##日のにっき(20×10)";
M;D:POKE &HFAFC,FNK OR 1:LINEINPUT R$:LS
ET A$=R$:PUT #1,A:GOTO 60
140 '●●● CTRL+STOP
150 CLOSE:POKE &HFAFC,FNK AND 254:PRINT
"おわり":END
```

リスト3の解説

【使い方】リスト2で作った日記帳に日記を書くためのプログラム。実行すると、「ひづけ(月，日)?」ときいてくる。書きたい日付、または読みたい日付を入力(12月25日なら、「12，25」とカンマで区切って入力)。日付の入力後、「よむ(R)¦かく(W)?」ときいてくるので、それぞれ読むならR、書くならWのキーを押す。読む場合は、その日付の日記を表示し、スペースキー入力待ち。書く場合は、自動的にローマ字かな入力モードになるので、ローマ字で日記を書く(かなキーを押せば英数字モードになる。ローマ字かな入力にもどるにはSHIFT+かな)。書くときは、1日のぶんが終わるまでリターンキーは押さないこと(リターンキーを押すと、そのときに書いてある内容をその日の日記として登録する)。ただし、1日のぶんは、それぞれ200字(10行)以内。

【プログラム解説】●行10〜20　初期設定／STOPキー割りこみ先の指定／STOPキー割りこみの許可／ユーザー定義関数の定義(BASICのワークエリア中にあるローマ字かな変換モードの切り換えスイッチの値)●行30〜50「DIARY.TXT」というファイルスペックで1レコード200バイトのランダムファイルを開く(バッファに割り当てられる変数はA$)●行60〜70　日付入力(M=月、D=日)／月日からそれに相当するレコード番号Aを計算●行80 ありえない日付をチェック(ただし、0月1日や12月40日はなくなるが、2月31日は見逃してしまう)●行90 英文字モードに切り換え／読むか書くかの選択／それに応じて、もどる、行100へ、行120への3つのうち1つへ飛ぶ●行100〜110　日記を読む(日付を表示／日付に応じたレコード番号Aにあるデータを表示／スペーキー入力待ち／キーバッファクリア／行60へ●行120〜130　日記を書く(日付を表示／ローマ字かな入力モードに切り換え／文字入力受け付け／バッファに転送／日付に応じたレコード番号にバッファにあ文字データを転送／行60へ)●行140〜150　STOPキー割りこみ処理サブ(ファイルを閉じ、英数字モードにもどして終了)

```
ひづけ(月,日)? 1,1
よむ(R)¦かく(W)? w
```

↑日付を入力したあと、自動的に英数字モードに切り換わるのでR(読む)かW(書く)を入力。ここでは、「w」を入力

```
1月 1日のにっき(20×10)
1992ねん と いえば バルセロナ
オリンピックだ。 ガウディのたて
もの や ECとうごう に わく ヨー
ロッパ を みにいきたい と おもう。
ヨーロッパ の ディズニーランド
にも いってみたいし。 ことしの 2ば
んめ の もくひょう は それにするだ
ももねー。 (あつし)
```

↑日付をかねたタイトルの下に日記を書く。1行20字なので10行まで。オーバーするとそのぶん無視される

ファイルでまわるダイヤモンド

BASICピクニック

#32 シーケンシャルファイルの実例

シーケンシャルファイルの、なにがシーケンシャルでどこがファイルなのか、目に見えないファイルの世界を見せるために、まわるダイアモンド再び。

MSX FAN 付録ディスク

→P116の「スーパー付録ディスクの使い方」参照

シーケンシャルファイル　sequential file　今月は、なぜか体調が悪く、つい文章が「ですます」調になってしまいます。そこで、ぜんぜん関係ないようですが、仮にもBASICピクニックを読んでくださっている読者のみなさまに、毎月タイトルバックのイラストがどのように制作されているかをお教えしましょう。このイラストをかいているのは、おはこんでおなじみのノミヤマ先生なのですが、わたしは毎月、BASICピクニックの企画が決まるとノミヤマ先生にファクスで今月やろうと思っていることを手紙にして送ります。今月は
「シーケンシャルファイルを使った3D線画アニメーションをやります。以前使った、まわるダイアモンドをシーケンシャルファイルを利用して高速に表示する、つもりです。イラストは、シーケンシャルファイルとダイアモンドから連想されるものに園児か小学生をまぶしてください。たとえば、全身にダイヤの装飾をつけたサナダ虫と遊ぶ子どもとか、クネクネと続く1本道のところどころに落ちているダイアモンドをひろう子どもたちとか」
という文章を送りました。正直いって、これでイラストをかいてもらえるのかどうか、毎月、不安なんです。それでも、今月もノミヤマ先生は上のようなイラストを仕上げてくれたのですが、ボツリと「今月は、ちょっとむずかしかったです」といってほほえまれました。というわけで、シーケンシャルファイルとは、このイラストのように、サナダ虫みたいなものです。

データをファイルに「PRINT」する

ちょうど1年まえの1991年2月号で、基本データをもとに毎回座標を計算しながらゆっくりとまわるダイアモンドの3D表示プログラムを掲載した。

この3Dダイアモンドは、①13個の頂点を②明るさのちがう30本の線分で結んで表示している。線分を結ぶ頂点の組み合わせはずっとおなじなので、毎回用意する必要があるのは、①で13組の座標データ(つまり26個の数値データ)、②で各線分ごとの明るさのデータ30個の、計56個のデータだ。

この56個のデータを、まとめて計算してシーケンシャルファイルにしてしまおう。

リスト1は、そのデータファイルを作るためのプログラムだ。じつは、これは1年まえのプログラムを見やすいように整理したうえで、ファイル書きこみの部分を追加するだけでできあがってしまった。

ようするに、座標などのデータを計算する端からシーケンシャルファイルに書きこんでいくだけなのだ。リストではわざと左寄り(左から3文字目)に書いている、

PRINT #1, ~

がシーケンシャルファイル「DIA-DATA」に書きこんでいる部分だ。シーケンシャルファイルへの書きこみは、画面やプリンタに文字を出力するときとほとんどおなじなのだ。

↑これ1個表示するには、頂点の組み合わせデータのほかに56個のデータが必要だ。リスト1は、とりあえずそのデータ群をディスクに書きこむためのプログラムだ

■リスト1　ダイアモンドのデータファイルを作る

DIA-MAKE.BP2

```
1000 '●●● MAKING DATA FILE
1010 COLOR 15,4,7:SCREEN 0:WIDTH 40
1020 CLEAR 1000:DEFINT A-Z
              :DEFSNG R,T,U,W-Z
1030 DF$="DIA-DATA"
1040 OPEN DF$ FOR OUTPUT AS #1
1050 '●●● FIRST SETTING
1060 R=1.7
1070 XM=0:YM=0:ZM=0
1080 U=ATN(1)/6
    :DIM T(2,2):T(0,0)=1:T(1,1)=COS(U)
    : T(2,2)=T(1,1):T(2,1)=SIN(U)
    : T(1,2)=-T(2,1)
    :DIM C(5):FORI=0TO1:FORJ=0TO2
    : C(J+I*3)=J:NEXT:NEXT
1090 '●●● DATA READING
1100 READ N,M,OX,OY
 :PRINT #1,N;M
1110 '● ARRAYS
1120 DIM X(N),Y(N),Z(N)
1130 DIM XX(N),YY(N),A(M),B(M)
1140 '● FIRST POSITION
1150 FOR I=0 TO N
    :  READ X(I),Y(I),Z(I)
    :  XM=XM+X(I):YM=YM+Y(I):ZM=ZM+Z(I)
    :NEXT
    :XM=XM/I:YM=YM/I:ZM=ZM/I
1160 FOR I=0 TO N
    :  X(I)=X(I)-XM
    :  Y(I)=Y(I)-YM
    :  Z(I)=Z(I)-ZM
    :NEXT
1170 '● A(m),B(m);START & END OF LINE
1180 FOR I=0 TO M
      :READ A(I),B(I)
 :PRINT #1,A(I);B(I);
    :NEXT
 :PRINT #1,
1190 GOTO 1300
1200 '●●● CALCULATING & PRINTING(#)
1210 READ A
1220 IF A=0 THEN END
1230 IF A=4 THEN SWAP T(2,1),T(1,2):GOTO
 1210
1240 A=A-1
1250 FOR I=0 TO N
1260    X=X(I)*T(C(A  ),C(A  ))
         +Y(I)*T(C(A  ),C(A+1))
         +Z(I)*T(C(A  ),C(A+2))
1270    Y=X(I)*T(C(A+1),C(A  ))
         +Y(I)*T(C(A+1),C(A+1))
         +Z(I)*T(C(A+1),C(A+2))
1280    Z=X(I)*T(C(A+2),C(A  ))
         +Y(I)*T(C(A+2),C(A+1))
         +Z(I)*T(C(A+2),C(A+2))
1290    X(I)=X
      :Y(I)=Y
      :Z(I)=Z
    :NEXT
1300
  PRINT #1,"XY";
1310 FOR I=0 TO N:
      :W=(Z(I)+500)/600
      :XX(I)=X(I)*R*W +OX
      :YY(I)=Y(I)  *W +OY
 :PRINT #1,USING"### ### ";XX(I),YY(I);
    :NEXT
1320
  PRINT #1,:PRINT #1,"L";
1330 FOR I=0 TO M
      :L=2^(4+(Z(A(I))<0)+(Z(B(I))<0)
        +(Z(A(I))+Z(B(I))<0))-1
 :PRINT #1,USING"##";L;
    :NEXT
 :PRINT #1,
1340 GOTO 1200
1350 '●●● N(ﾁｮｳﾃﾝ)   M(ﾍﾝ)     OX, OY
1360 DATA 12,        29,       256,80
1370 '●●●  X(n),Y(n),Z(n)
1380 DATA   44.282, 0      ,123.301
1390 DATA   19.282, 0      , 80
1400 DATA   44.282, 0      , 36.6988
1410 DATA   94.282, 0      , 36.6988
1420 DATA  119.282, 0      , 80
1430 DATA   94.282, 0      ,123.301
1440 DATA  69.282, 22.9825,160
1450 DATA  0      , 22.9825,120
1460 DATA  0      , 22.9825, 40
1470 DATA  69.282, 22.9825,  0
1480 DATA 138.564, 22.9825, 40
1490 DATA 138.564, 22.9825,120
1500 DATA  69.282,102.983 , 80
1510 '●●● A(m),B(m)
1520 DATA 0,1,1,2,2,3,3,4,4,5,5,0
1530 DATA 6,7,7,8,8,9,9,10,10,11,11,6
1540 DATA 0,6,0,7,1,7,1,8,2,8,2,9,3,9,3,
10,4,10,4,11,5,11,5,6
1550 DATA 6,12,7,12,8,12,9,12,10,12,11,1
2
1560 '●●● A:MAWALING DATA
1570 DATA 1,1,1,2,3,3,3,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,3,3,3,4,1,1,1,1,
2,4,1,1,1,1,1,1,1,2,3,3,3,3,3,3,0
```

リストのプログラム解説

1000～1040　画面などの初期設定／データファイルを「DIA-DATA」としてオープン

1050～1080　変数、配列の初期化および設定　●R=SCREEN7用の縦横比修正用データ●XM、YM、ZM=仮想3次元座標系でのダイアモンドの中心座標(回転の中心)●U=7.5度のラジアン値(1回に回転する角度)●T(n, m)=回転座標の計算(行1250～)で使用する係数●C(n)=配列Tを各座標軸で使いわける引数管理用

1090～1180　初期データ読み出し(行1350～1550から読み出し)●N=頂点数−1●M=辺の数−1●OX、OY=画面の座標系での中心座標●X(n)、Y(n)、Z(n)=仮想3次元座標系での頂点nの座標●XX(n)、YY(n)=画面の座標系での頂点nの座標●A(m)、B(m)=辺mの始点、終点の頂点番号※行1100でN、Mを、行1180でA(m)、B(m)のデータファイルへの書きこみ

1190　行1300に飛ぶ

1200～1340　回転図形の座標計算とデータファイルへの書きこみ

1210～1240　回転方向データAの読み出しと評価(A：0=終了、1=X軸回転、2=Z軸回転、3=Y軸回転、4=逆回転。1～3は−1して配列Cの添字とする)

1250～1290　仮想3次元座標系での回転計算　●X、Y、Z=計算された回転後の座標一時保存用

1300～1310　表示用座標データの計算とファイルへの書きこみ　●W=画面上の奥行き表現用

1320～1330　辺mの明るさLの計算とファイルへの書きこみ　●L=3段階の明るさに対応したパレットコード

1340　行1200に飛ぶ

1350～1570　データ(内容はREM文に書かれた変数名を参照)

※行1180、1330の末尾にある「PRINT#1,」は改行用

※リスト1は、付録ディスクに収録されている「DIA-MAKE.BP2」のREM文などを一部修正していますが、動作はまったくおなじです。

前ページの続き　おはこんのタイトルバックも同様の手法でノミヤマ先生にかいていただいています。今月は、「2月というとセブンイレブンとかバレンタインとかですが、今年は2月11日建国記念日(紀元節)か2月15日仏滅記念日(注・そんなものありません)のどちらかで。日向の地に降臨したニニギノ命によりそうババか、北枕で地磁気と共鳴しながら入滅していくシャカにとりすがるババか、ババ顔した大豆によるメンデルの法則図とかでもいいです」
と書いて送ったら、89ページのようなイラストになりました。

リスト2の使い方　「DIA－DATA」というファイルは、すでに付録ディスクのなかに入っているので、リスト2「DIA－DSPL. BP2」をRUNすれば、このページ下でくるくるしているダイアモンドがディスプレイに出てくるだろう。ただ、いまのままではディスクからデータを読み出すたびに動きがちょっと止まってしまう。そこで、行3000以降に、MSX2のRAMディスク(CALL MEMINIの互換RAMディスク)を利用するルーチンを付けておいた。RUNのかわりに、RUN3000を実行すると、RAMディスクを設定し、付録ディスクの「DIA－DATA」をRAMディスクに移しかえて、DF$を「MEM:DIA－DATA」として、行2050に飛ぶ。これ以降は、RAMディスクのデータにもとづいてまわることになる。互換RAMディスクの使い道はこういうところにあったのだ!

EOF(ファイル番号)　EOF関数は、その番号のシーケンシャルファイルの最後のレコードを読み出したかどうかを教えてくれる。たとえば、ファイル#1の最後のデータを読み出していると、EOF(1)は－1。このとき、さらに読みだそうとすると、
Input past end
というエラーが出る。

ファイルからデータを「INPUT」する

前ページのリスト1を実行すると、そのときセットしていたディスクに「DIA－DATA」というデータファイルを作り、図1のようにデータを書きこんでいく。

■シーケンシャルファイルに並べられていくデータ

図1は、1マスが1バイトで見た目どおりの「文字」が書きこまれていくことを意味している。たとえば、このファイルの先頭には、前ページ、リスト1の行1100にある、
PRINT #1, N;M
によって、12(=N)と29(=M)という数値が書きこまれるのだが、データファイル「DIA－DATA」のそれに対応する部分は、図1のように、
空白、1、2、空白2個、2、9、空白、CR、LF
というふうにデータがならんでいる。「1」は数値の1ではなく、文字としての数字つまりCHR$(49)の1だ。

これは、ふつうのPRINT文でもまったくおなじだ。

リスト1の行1100を
PRINT　　N;M
とした場合、やはり、
12　29
と表示して、カーソルを次の行の先頭に持っていく。この場合も、それぞれのキャラクタコードをVRAMに書きこんでいるわけだから。また、もう1つ、注目したいのは、
「次の行の」
「先頭に持っていく」
という動作だ。これには、CRコードとLFコードが作用している。文章表現上、順番は前後しているが、前者がLFコード、後者がCRコードの作用だ。

じっさいには、CRコードによってカーソルはその行の先頭にもどり、LFコードによって次の行へ送られる。

PRINT文に続く式(数や文字など)の末尾になにもないと、CRとLFが自動的に追加され、「;」を区切りに使ったり、末尾に付けたりしておくとCRもLFもなく、次にPRINTされるものが続けて表示される。「,」も決められた表示位置に頭ぞろえで表示されるという点以外は同様だ。そのほか、USING文もおなじように使える(リスト1では後述する理由で、USINGも使った)。

前ページのリスト1の「PRINT #1,」を、「PRINT」に置き換えれば、データファイル「DIA－DATA」に書きこむのとまったくおなじ形式で画面上にデータを表示してくれる。

■シーケンシャルファイルからデータを取り出す

ようするに、「DIA－DATA」には、図2のようにデータが書きこまれているのである。

このデータファイルを、リスト2の行2050のように「FOR INPUT」(入力用ファイルの指定)でオープンすると、以降、たとえば、
INPUT#1, A
とするたびに、変数Aに最初から順に数値データが入っていくのだ。まず、12が入り、次に
INPUT#1, A
とすると、こんどは29が入り、次は0、1、1、……。このように空白を区切りとして、数値データを切り出してくれる。

では、ダイアモンドを表示するときに必要なデータを順番に書きこんでおき、その順に取り

■図1　シーケンシャルファイルのデータの並び方

	1	2			2	9	CR	LF		Ø			～

～	CR	LF	X	Y	2	1	8			6	3		～		CR	LF	EOF

CR＝CHR$(13)　LF＝CHR$(1Ø)　EOF＝CHR$(26)

●シーケンシャルファイルには、CR、LF、EOFの3つのコードが付きものだ。●CRとLFは、2つ1組で、レコードの区切りを表す。シーケンシャルファイルにおけるレコードとは、単純にいうとLINE INPUT#で一度に読み出されるデータのまとまりの単位だといえる。たとえば、アスキーセーブされたプログラムの1つの行は、まさに典型的なレコードだ。ちなみに、CRはCTRL＋M(またはRETURNキー)、LFはCTRL＋Jに対応するコントロールコードだ。●EOFは、End Of File(ファイルの終わり)の略で、CTRL＋Z。シーケンシャルファイルの終わりにはかならずこのコードが付く。

↑くるくるまわるダイアモンド。1年まえより速くなった

出して表示していけばかんたんだ。そこで、ごくストレートにデータファイルを作り、それをもとに表示させてみたのだが、うまくいかなかった。なぜなら、その方法だと1つ1つ計算してやっていくよりも、格段にスピードが遅かったのである／ ガーン。ファイルを読み出すのはけっこう時間がかかるのだ。

そこで、一計を案じて、リスト1でデータファイルを作るときに、「XY」などの文字を入れたり、USING文を使ったりして形を整えておき、リスト2では、最初のN、Mと配列A、Bだけを数値データとして単純に読みこみ、以降は、文字変数XY$、L$で読み出す（行2150、行2170）ことにした。

シーケンシャルファイルでは、文字データの区切りは、CHR$(44)（コンマ）かCR+LFなのだ。今回のデータファイルでは、コンマをまったく使っていないので、一度、文字変数としてデータを読みこむと、その変数のなかにCR+LFまでのすべてのデータが入ってしまう。今回は使わなかったが、LINE INPUT#によって読み出すと、コンマも含めてまるごと変数に入れてくれる。

リスト2は、そうやってデータで腹いっぱいになったXY$やL$を、あらかじめ決めておいた文字数で区切って切り出し、数値に変換して使っている、というわけなのだ。

こうすると、ファイルから読み出すのは1つのダイアモンドにつき、2回ですみ、あとは単純な処理なので、それほど時間がかからない。

データが多いとき、シーケンシャルファイルは、数値で書きこみ、文字で取り出すのがよさそうだ。

■図2 「DIA-DATA」の中身

```
 12   29                                           ―N、M
 0  1  1  2  2  3  3  4  4  5  5  0  6            ┐A(0)、B(0)
7  7  8  8  9  9  10  10  11  11  6  0            │
6  0  7  1  7  1  8  2  8  2  9  3  9  3          │ ∫
  10  4  10  4  11  5  11  5  6  6  12            │
7  12  8  12  9  12  10  12  11  12               ┘A(M)、B(M)
XY218   63 185   65 224   66 288   66 327   6     ┐
5 294   63 256   84 150   84 166   83 256   8     │XY$
3 346   83 362   84 256 150                       ┘
L15 3 1 3151515 7 1 1 715151515 3 1 1 1           ┐L$
1 315151515 3 3 315                               ┘
XY218   58 186   65 224   70 288   70 326   6     ┐
5 294   58 256   74 150   79 166   87 256   9     │XY$
0 346   87 362   79 256 151                       ┘
L 7 1 1 1 71515 7 1 1 7151515 7 1 1 1 1           ┐L$
1 1 715151515 3 3 315                             ┘
XY218   54 186   65 224   75 288   75 326   6     ┐
5 294   54 256   64 150   75 165   91 256   9     │XY$
8 347   91 362   75 256 151                       ┘
L 7 1 1 1 71515 7 1 1 7151515 7 1 1 1 1           ┐L$
1 1 715151515 3 3 315                             ┘
XY218   50 186   66 224   80 288   80 326   6     ┐
6 294   50 256   55 150   70 165   95 256  10     │XY$
5 347   95 362   70 256 149                       ┘
L 7 1 1 1 71515 7 1 1 7151515 7 1 1 1 1           ┐L$
1 1 715151515 3 3 315                             ┘
XY225      190   61 224      288   82 328
1 30        62   55 153        2   88 251
5 3          3   78 24
             15 7                 1 1
```

■リスト2　ダイアモンドのアニメーション

DIA-DSPL.BP2

```
2000 '●●● DRAWING
2010 '     FIRST SETTING
2020 COLOR 15,4,7:SCREEN 0:WIDTH 40
2030 CLEAR 1000:DEFINT A-Z
2040 DF$="DIA-DATA"
2050 OPEN DF$ FOR INPUT AS #1
2060 INPUT #1,N,M
2070 DIM A(M),B(M),X(N),Y(N)
2080 FOR I=0 TO M
     :  INPUT #1,A(I),B(I)
     :NEXT
2090 '●●● SCREEN SETTING
2100 COLOR 15,0,4:SCREEN 7
2110 P=0:Q=1:SETPAGE ,1:SCREEN,0
2120 COLOR=(4,1,1,2)
     :COLOR=(1,3,3,3)
     :COLOR=(3,5,5,5)
     :COLOR=(7,6,6,6)
2130 '●●● DRAWING A DIAMOND
2140 SETPAGE P,Q
     :CLS
     :SWAP P,Q
2150 INPUT #1,XY$
2160 FOR I=0 TO N
     :  X(I)=VAL(MID$(XY$,8*I+3,4))
     :  Y(I)=VAL(MID$(XY$,8*I+7,4))
     :NEXT
2170 INPUT #1,L$
2180 FOR I=0 TO M
     :  L=VAL(MID$(L$,2*I+2,2))
     :  LINE (X(A(I)),Y(A(I)))
           -(X(B(I)),Y(B(I))),L,,OR
     :NEXT
2190 IF EOF(1) THEN
       CLOSE
     :  OPEN DF$ FOR INPUT AS #1
     :  INPUT #1,N$,N$
2200 GOTO 2130
3000 '●●●
3010 '●●● FLOPPY >>> RAM
3020 CLEAR 1000:MAXFILES=2
3030 DF$="DIA-DATA"
3040 OPEN DF$        FOR INPUT  AS #1
3050 CALL MEMINI
     :OPEN "MEM:"+DF$ FOR OUTPUT AS #2
3060 INPUT #1,A$:PRINT #2,A$
3070 IF EOF(1) THEN
       CLOSE
     :  DF$="MEM:"+DF$:GOTO 2050
3080 GOTO 3060
```

シーケンシャルファイル最後の戦い

BASICテクニック

#33 ファイルに関する命令

ファイルに関する命令や関数はじつにたくさんある。そのうちから、とりあえずシーケンシャルファイル用のものだけに限定して紹介する。

テキスト画面、グラフィック画面、プリンタ　この3つのデバイス(入出力機器のこと。ここではファイルにまつわる入出力先のこと)では、ファイル名とモード指定(FOR OUTPUT)を省略するのがふつうだ。ファイルをオープンするときは、

OPEN"〈デバイス名〉:" AS #〈番号〉

だけでいいわけだ。ファイル名がいらないのは使いわける意味がないからで、モード指定がいらないのは出力モードしか使えないからだ。この3つのデバイスではファイルへの出力は、すなわち文字の表示、印字になる。

なお、デバイス名は以下のとおり。

グラフィック画面……GRP:
テキスト画面…………CRT:
プリンタ………………LPT:

ファイル番号　ファイルをオープンするときの末尾に来るのがこれ。通常、1しか使えないが、2つ以上のファイルをオープンしたいときに意味を持ってくる。その場合、あらかじめ、

MAXFILES=n

で必要な数だけファイル番号を使えるように設定しておかなくてはいけない(nは必要な数)。

たとえば、上記の命令でnを2としていた場合は、ファイル番号#1と#2が使え、それぞれに別々のファイルをオープンできるというわけだ。

シーケンシャルファイルの3つのモード

今回は、ファイル操作に関するBASIC知識を総まとめにしておこうと思ったが、シーケンシャルファイルに関するものだけでページが埋まってしまった。ランダムアクセスファイルについては、三択クイズジェネレータのサンプルとともに次回やることにする。

†

シーケンシャルファイルは、いろんなところでオープンできる。ディスクドライブはもちろん、MSX2の影のRAMディスク、ターボR(DOS2)の真のRAMディスク、カセットテープ(ただしターボRは使用不可)、**テキスト画面**、**グラフィック画面**、**プリンタ**、とこんなにある。

それに対して、ランダムアクセスファイルは、ディスクドライブと真のRAMディスクだけ。真のRAMディスクは、BASICの命令上、ふつうのディスクドライブとおなじ種類のもの(ドライブ番号Hとして設定)なので、実質的にはランダムアクセスファイルはディスクドライブでしか使えないといえる。

このことは、この2つのファイル形式の複雑さを物語っている。いうまでもなく、いろんなところで使えるシーケンシャルファイルのほうが圧倒的に単純なのである。

その単純なシーケンシャルファイルで、唯一、複雑かもしれないのが3つのモードの存在だ。

†

ディスクでシーケンシャルファイルを扱うときは、オープンするときにかならずモードを指定しなければならない。モードを指定せずにオープンしてしまうと、自動的にそれはランダムアクセスファイルになる。こちらは命令も関数もまったくちがうので、わけがわからなくなってしまう。

シーケンシャルファイルのモードには以下の3つがある。

①出力(OUTPUT)
②入力(INPUT)
③追加(APPEND)

出力とは、MSX本体からファイルへ出力するという意味で、つまり、書きこむこと。このモードでオープンすると、そのとき指定したファイル名の中身はかならずカラッポになる。

入力とは、出力の逆ですでに存在するファイルからデータを取り出すこと。

追加とは、すでに存在するファイルにデータを付け足すこと。

まず出力モードでファイルを作ろう。右ページ図1に従って命令を打ちこんでいけば、それぞれ右図に示すような構造のデータファイルができあがっていく。

●図1 あるシーケンシャルファイルの誕生

```
OPEN "TEST-SEQ.DAT" FOR OUTPUT AS #1
```

この命令で「TEST−SEQ. DAT」という名前のファイルが出力用に**ファイル番号**＃１でオープンされる。最初はカラッポ。今後、このファイルは「＃１」という番号で呼ばれることになる。

TEST-SEQ. DAT

```
[ | | | | | | | | | ]
```

CR …………CHR$(13)
LF …………CHR$(10)
EOF…………CHR$(26)

```
A$="BASIC":B$="PICNIC"
PRINT #1,A$;B$
```

文字をいろんな形で出力してみよう。「PRINT ＃１,」は＃１のファイルに出力するということだ。A$、B$をセミコロン(；)でつないで出力すると、ファイルには右のように続けて記録される。

TEST-SEQ. DAT

```
BASICPICNIC CR LF
```

```
PRINT #1,A$,B$
```

上のセミコロンのかわりにコンマ(,)でつないで出力。すると、出力した文字列のあいだに10文字ぶんの空白を置いて記録される。以上の2つのことは、「＃１,」を削除して、たんなるPRINT文で画面上に出力してもまったくおなじように表示される。これにかぎらず、出力に関しては基本的にPRINT文とまったくおなじように、セミコロン、コンマ、USING、文字式などが使える。

TEST-SEQ. DAT

```
BASICPICNIC CR LF
BASIC          PICNIC CR LF
```

```
PRINT #1,A$;",";B$
```

これの意味は次の32ページの図2を見てからでないとわからないが、ファイル上にコンマがほしいときは、このようにダブルクォート(")でくくったコンマをあいだにはさむ。すると、当然だが、右のようにDATA文のなかでデータがコンマで仕切られているように2つの文字列が仕切られて記録される。これが入力モードのときにたいせつな意味を持つ。

TEST-SEQ. DAT

```
BASICPICNIC CR LF
BASIC          PICNIC CR LF
BASIC,PICNIC CR LF
```

```
PRINT #1,1;2,123
```

こんどは数値を出力してみよう。ファイル上ではすべてが文字の形で記録される。１という数値を出力すると、前後に空白を加えた文字列「□1□」(□は空白)を記録する。セミコロン、コンマの意味は上とおなじ。数値も文字列に変えて記録するということは、とうぜんあとから文字列として読み出すこともできるということだ。この逆、つまり文字列を数値として読み出そうとしてもREAD文、DATA文のようにエラーにはならない。ただ、数値の形になっていない文字列はつねに0として扱われる。

TEST-SEQ. DAT

```
BASICPICNIC CR LF
BASIC          PICNIC CR LF
BASIC,PICNIC CR LF
 1  2          123  CR LF
```

```
CLOSE
```

CLOSEは、ファイルを閉じる命令。ファイルが複数オープンされているときは、このままで「全ファイルを閉じる」という意味になり、「CLOSE　＃１」というふうにファイル番号を指定して、そのファイルだけを閉じることもできる。この命令までは、これまで出力していたデータはメモリ内にためられているだけでディスクには書きこまれていない。ここで「CLOSE」リターンと入力すると、ガッガッとディスクが動いて、図のようなデータを持った「TEST−SEQ.DAT」というファイルができあがるのだ。シーケンシャルファイルの場合、ファイルの末尾のしるしとして「EOF」というコードが付け加えられる。

TEST-SEQ. DAT

```
BASICPICNIC CR LF
BASIC          PICNIC CR LF
BASIC,PICNIC CR LF
 1  2          123  CR LF EOF
```

いーかげんにしろ　こういわれないために、関数EOFというのがある。33ページのリスト行60でも使っている。この関数は

EOF(n)　※nはファイル番号

という形でシーケンシャルファイルにかぎって使用する。#nのシーケンシャルファイルにまだ入力用の命令によって読み出すことのできるデータが残っていれば、この関数は0になっており、最後のデータが読み出された時点で−1に変わる。だから、この関数が−1になっているかどうかさえ調べておき、−1になったら入力をやめるようにすれば、Input past endは見なくてすむわけだ。

INPUT$(n, #1)　この関数もほかで見かけたことがあるだろう。たんにINPUT$(n)とすれば、キーボードからの入力を待つ関数になる。この入力用関数はなんでも読みこむのでおもしろいが、たった1つ、EOFだけは読みこめない。

取り出し方も4とおりある

CR、LFで区切られたデータのブロックを「レコード」という。BASICプログラムの「行」も、このレコードの一種だ。じっさい、プログラムをアスキーセーブすると、それは典型的なシーケンシャルファイルになっていて、行のおわりにはCRとLFがかならずくっついている。興味のある人は、アスキーセーブしたファイルからここで述べる方法でデータを読み出してみればよくわかる。

†

シーケンシャルファイルを読み出すには、まず入力モードでファイルをオープンする。

OPEN "TEST-SEQ.DAT" FOR INPUT AS #1

そして、

LINEINPUT #1,A$

を実行し、続いて

PRINT A$

を実行すると

BASICPICNIC

と表示されるだろう。もう1回、

LINEINPUT #1,A$

を実行し、同様にA$を表示してみると、

BASIC(空白10個)PICNIC

またおなじことをすると、

BASIC, PICNIC

もう1回やると、

1　2(空白12個)123

さらにやると、

Input past end

というエラーが出る。ファイルにはもう出せるデータはないという意味で、ひらたくいえば、

いーかげんにしろ

といっているわけだ。

シーケンシャルファイルは、入力の命令が実行されるたびに頭から順にデータを渡し、けっしてあともどりはしない。

CLOSE

を実行し、また入力モードでオープンすれば、ふたたび頭から取り出せるようになる。次は、

INPUT #1, A$
INPUT #1, A

のそれぞれでおなじことをやってみよう。図2の青い四角は各命令で1回ごとに読み出される区画を示したものだ。ただし、数値の形になっていない文字を数値変数で読み出すと「0」として扱う。

●図2 4とおりの「インプット」

TEST-SEQ. DAT

BASICPICNIC CR LF
LINEINPUT #1,A$
INPUT #1,A$
INPUT #1,A
A$=INPUT$(1,#1)

BASIC　　　　　　　　　　PICNIC CR LF
LINEINPUT #1,A$
INPUT #1,A$
INPUT #1,A
A$=INPUT$(1,#1)

BASIC,PICNIC CR LF
LINEINPUT #1,A$
INPUT #1,A$
INPUT #1,A
A$=INPUT$(1,#1)

1　2　　　　　　　　　　　　123 CR LF EOF
LINEINPUT #1,A$
INPUT #1,A$
INPUT #1,A
A$=INPUT$(1,#1)

LINEINPUT #1, A$(文字変数)　CRの手前までのデータを読みこむ
INPUT #1, A$(文字変数)　CRまたはコンマの手前までのデータを読みこむ
INPUT #1, A(数値変数)　CR、コンマ、空白の手前までのデータを数値として読みこむ。数値の形でない場合は0とする
A$=INPUT$(n, #1)　ファイル番号#1からnバイト読みこんでA$に入れる。CRもLFもほかのデータと同様に扱う
※図の青い棒グラフはそれぞれの入力命令が1回行われるたびに読みこまれるデータ量のイメージ

◉図3 シーケンシャルファイルの増殖

```
OPEN "TEST-SEQ.DAT" FOR APPEND AS #1
```

モード指定のところで「FOR APPEND」としてファイルをオープンすると、そのファイルにデータを追加できるようになる。ただし、指定するファイルはかならずすでに作ってあるファイルであること。指定されたファイルがディスクのなかにないと「File not found」のエラーが出る。追加されるデータは、そのファイルを頭から見ていって最初に出会ったEOFの位置から上書きされていく。

TEST-SEQ. DAT

B	A	S	I	C	P	I	C	N	I	C	CR	LF

BASIC　　　　　　　　　　PICNIC CR LF

BASIC,PICNIC CR LF

　1　2　　　　　　　　　　　123　CR LF EOF

```
PRINT #1,-1;-2;
CLOSE
```

データを追加してファイルを閉じると、新たなEOFコードを末尾に付け加えてファイルの更新が終わる。ちなみに、ここでは出力データの最後にわざとセミコロン(;)を付けてみると、予想どおり、CR、LFのどちらもファイルには書きこまれず、いきなりEOFで終わっている。次回、またAPPENDするときは、このEOFから書きこまれていくので、－1、－2とおなじレコード内に追加データが入っていくことになる。また、負の数値の収まり方にも注目しておこう。

TEST-SEQ. DAT

BASICPICNIC CR LF

BASIC　　　　　　　　　　PICNIC CR LF

BASIC,PICNIC CR LF

　1　2　　　　　　　　　　　123　CR LF

－1　－2　EOF

このほかに、もう1タイプ、INPUT$(バイト数、ファイル番号)という強力な関数がある。これは、指定ファイルから指定バイト数だけ読み出すというものだが、これだけはレコードの区切り、つまりCR、LFにこだわらずに読んでいく。

図2を見ると、A$＝INPUT$(1, #1)のところだけ、CR、LFすらも読んでいることがわかる。

読み出すたびにバイト数を変えてもいいのだから、いろんな使い方ができそうだ。

†

シーケンシャルファイルは部分的に変更したり削除したりできないが、データの追加だけはできる。図3がその手順だ。ここでは、図1で作ったファイルに、－1と－2を加えた。そうして作ったファイルをまた、追加モードでオープンして、データを追加することができる。

ところで、ファイルの実験のときに、ファイルの用事がすんだらこまめにCLOSEすること。プログラムをとちゅうで止めたりしたときも同様だ。そうでないとディスクの中身が壊れる可能性があるので要注意。

◉リスト シーケンシャルファイルの中身を実感

```
10 COLOR 15,4,7:SCREEN 1:WIDTH 29:KEYOFF
20 VPOKE &H201F,&H9
30 FOR I=0 TO 23
   : LOCATE 0,0
   : PRINT CHR$(27)+"L";STRING$(29,254);
   :NEXT
40 LOCATE 0,0
50 OPEN "TEST-SEQ.DAT" FOR INPUT AS #1
60 IF EOF(1) THEN BEEP:GOTO 90
70 PRINT INPUT$(1,#1);
80 GOTO 60
90 FOR I=0 TO 1:I=-STRIG(0):NEXT
100 SCREEN 0:END
```

※このプログラムは付録ディスクには収録されていません。打ちこんで実行してください。

プログラム解説

【使い方】行50の「TEST－SEQ. DAT」を好きなファイル名に置き換えて実行すると、そのファイルの中身を空白や改行を含めて表示する。

【解説】●行10＝画面初期設定 ●行20＝キャラクタコード254を明るい赤の空白にする ●行30＝画面を赤い空白で埋める(エスケープシーケンスを使用) ●行40＝カーソルを(0, 0)に ●行50＝入力モードでファイルを＃1としてオープン ●行60＝ファイルの終わりならビープ音を鳴らして行90へ飛ぶ ●行70＝＃1から1バイト読みこみ(前の文字に続けて) 表示する ●行80＝行60へ飛ぶ ●行90＝スペースキー入力待ち ●行100＝画面をスクリーン0にして終了

BASICPICNIC
BASIC　　　PICNIC
BASIC,PICNIC
 1 2　　　　123
-1 -2

⬆赤い部分にはなにもないと考えてほしい。そこには記憶する場所すらない。ファイルの中はこんな感じなのだ

超イージーな3択クイズツール

BASICテクニック

#34 3択クイズとランダムファイル

運命の分かれ道で2つに1つの選択を迫られる状況は、3択クイズに似ている。Q 右、左どちらの道を行きますか? ①右の道 ②左の道 ③引き返す

付録ディスク
→P.118の「スーパー付録ディスクの使い方」参照

リスト1の緑地部分 ここは引き出して解説している部分以外の、ファイルに関する部分だ。
行170にある
LOF(n)
はファイル番号#nの大きさをバイト単位で教えてくれる関数。行170ではすでに問題ファイルを作っていた場合に何問ぶん(何レコードぶん)あるかを計算するために使っている。
行230の
GET#1, R
は33ページ参照。

なぜランダムファイルなのか

3択クイズのためには
①問題を作る
②出題する
の2段階が必要だ。

①の部分のためのプログラムが、右ページのリスト1だ。ランダムファイルの1つのレコードのなかに問題文と3つの選択肢、答え、ヒントの6種類のデータを収めることにした。

なぜ、ランダムファイルなのか。ランダムファイルは内容を更新することがかんたんにできるからだ。クイズの問題にかぎらず、あるていどの長さを持つ文章は、作ったあとで誤りや誤字脱字が見つかるものだ。だから、あとから一部修正する機能はどうしてもなくてはならない。

シーケンシャルファイルは、「追加」ができても一部修正はできないので、最初からクイズのデータを収めるファイルとしては失格なのだ。

リスト1のプログラム解説

【適応機種】MSX2+、ターボR。または、漢字BASIC(CALL KANJI)を拡張したMSX2。
【準備】付録ディスクから
MAKEDATA. BP4
というファイルを自分のディスクにコピーし、31ページ下の「ごめんなさい」に従って修正し、おなじファイル名でSAVEしなおす。そのディスクを書きこみ可にしてディスクドライブにセットしておく。
【使い方】実行すると、最初に「登録先のレコード番号」(レコード番号は問題番号とおなじ)を表示する。はじめてのときは「1」、すでに作っているファイルがあるときは、そのファイルの問題数+1が表示される(ディスクをセットしていなかったり、書きこみ不可にしていたりするとエラーになる)。カーソルキーの上下で数値が増減する。リターンキーを押すと、そのレコードにすでに問題があれば更新、なければ新規の作成になる。全角文字(漢字)の使用を前提に作っているが、半角文字を混ぜて使うこともできる(CTRL+スペースキーで漢字入力の切り換えになるが、くわしくは各機種の取扱説明書を参照)。各項目(問題、選択肢3つ、答え、ヒント)を入力していき、1問終わるたびにスペースキーで継続、ESCキーで終了する。継続すると、最初にもどり、そのファイルの最大レコード番号+1の数値が表示される。また、CTRL+STOPで中断したときは、CLOSEまたはENDをダイレクトに実行しておくのを忘れないように。

```
問題入力(3行以内)*********?
MSX・FANの創刊号は1987年3月8日
に発売されました。では、今月号でMファン
は

何執念でしょう?
```

↑リスト1のプログラムの実行画面。1問目の問題を入力中(AISTでの画面)

←ホントは3月6日ですけど(北根)。すみません(山科)

※日本語入力の機能は、機種によって異なるため、写真のように入力できない機種もあります

◉リスト1　3択クイズのデータファイルを作る

MAKEQUIZ.BP4　　+α

```
100     'SAVE"MAKEQUIZ.BP4
110 CLEAR 1000:DEFINT A-Z:DEFUSR=342
120 COLOR 15,4,4:SCREEN 0:KEYOFF
130 CALL KANJI1:WIDTH 38
140 RESTORE 130:FOR I=0 TO 5:READ L(I)
:NEXT:DATA 112,34,34,34,1,36
150 OPEN "QUIZ.DAT" AS #1 LEN=256
160 FIELD #1,L(0) AS M$(0),L(1) AS M$(
1),L(2) AS M$(2),L(3) AS M$(3),L(4) AS
 M$(4),L(5) AS M$(5)
170 LR=LOF(1)/256:R=LR+1:A$=""
180 CALL CLS:A=USR(0)
   :PRINT "登録先のレコード番号は  ";
190 IF INKEY$=CHR$(13) THEN 230
200 LOCATE 22,0:PRINT USING"####";R;
210 S=STICK(0):IF S=0 THEN 190
215 R=R+(S=1)-(S=5):IF R<1 THEN R=1
220 BEEP:GOTO 190
230 IF R<=LR THEN GET #1,R
    ELSE FOR I=0 TO 5
       :LSET M$(I)="":NEXT
240 CALL CLS:A=USR(0)
   :PRINT CHR$(10);M$(0);CHR$(11);
250 INPUT "問題入力（３行以内）＊＊＊
＊＊＊＊＊";A$
260 LSET M$(0)=LEFT$(A$,L(0))
270 FOR I=1 TO 3:CALL CLS
280   PRINT CHR$(10);M$(I);CHR$(11);
            MID$("１２３",I*2-1,2);
290   INPUT "番の選択肢入力（全角17字
以内）＊＊";A$
300   LSET M$(I)=LEFT$(A$,L(I))
310 NEXT
320 CALL CLS:A$=M$(4)
330  PRINT "正解入力    ";A$
    :INPUT "正解は何番ですか";A$
    :CALL KACNV(A$,A$):A=VAL(A$)
    :IF A<1 OR A>3 THEN 320
340 LSET M$(4)=HEX$(A)
350 CALL CLS:A=USR(0)
   :PRINT CHR$(10);M$(5);CHR$(11);
360 INPUT "ヒント入力（全角19字以内）
＊＊＊＊＊";A$
370 RSET M$(5)=LEFT$(A$,L(5))
380 CALL CLS
   :PRINT "登録します"
390 PUT #1,R
400 PRINT CHR$(10);"レコード番号";R;"
の問題を処理しました"
410 PRINT CHR$(10);"もっと作るならスペ
ースキー、やめるならＥＳＣキーを押して
ください"
420 IF STRIG(0) THEN R=R+1:GOTO 170
430 IF PEEK(&HFBEC) XOR 251 THEN 420
440 A=USR(0):END
```

ランダムファイルのOPEN

OPEN〈ファイル名〉AS ♯n LEN=〈バッファの長さ〉

ランダムファイルのOPENは、シーケンシャルファイルとちがってモード指定がない。
逆に、モード指定をせずにOPENしたディスク上のファイルは、かならずランダムファイルになる。
「AS ♯n」のnはファイル番号。その直後に「LEN=〜」の書式で、バッファ（ランダム入出力バッファ）の長さをバイト単位で指定する（最大256）。これは省略可能で、省略した場合は256に設定される。
バッファとはディスクとデータのやりとりをする中継地点となるメモリの一部分で、バッファの長さは、レコードの長さだ。

ランダム入出力バッファを仕切る

FIELD ♯n,〈文字変数〉AS〈文字変数〉,……

バッファにデータを書きこんだり、バッファからデータを読み出したりするときは、そのバッファ用に特別に割り当てた文字変数を使う。この場合は、M$(0)〜M$(5)で、バッファとのやりとり以外でこの変数を使うと正常な動作をしなくなるので注意。
このバッファ用文字変数を割り当てる命令が上記の命令だ。行160のステートメントは、「左からかぞえてL(0)バイトをM$(0)に、L(1)バイトをM$(1)に……」というふうに左から順々に各文字変数用にランダム入出力バッファを仕切っているのだ。

データをバッファに転送する

LSET〈文字変数〉=〈文字式〉

RSET〈文字変数〉=〈文字式〉

ランダムファイルにデータを送るには、まずバッファに書きこまなくてはいけない。バッファに書きこむということは、すなわち、バッファ用文字変数にデータを代入するということだ。しかし、バッファ用文字変数をふつうの代入文で一度でも使用すると、バッファ用文字変数であるのをやめてしまうのだ。
そこで、上記の2つの命令がある。LSETは左詰め、RSETは右詰めに代入する。ここではヒントのみRSETにした。

バッファのデータをファイルに書きこむ

PUT ♯n,〈レコード番号〉

バッファ用文字変数に、LSET、RSETを使って必要なデータを渡すと、メモリ上のランダム入出力バッファにFIELD文で指示したとおりの書式でデータがセットされる。このバッファ上のデータをディスク上のランダムファイルに送りこまなければ、そのデータはファイルのものにはならない。
バッファからファイルへ、データを送りこむ命令が、PUT命令だ。この命令を実行すると、その時点でバッファに入っていたデータを指定したレコード番号でファイルに書きこむ。このとき、ディスクがガッガッと動き、ディスクランプが点灯する。

ファイルを閉じて終了する

END、CLOSE

OPENしたファイルは、用がすんだらCLOSEすること。これをおこたると、思いがけないときに書きこみがあるなど、ディスクを壊しかねない。
そこで、プログラムの終わりにはEND、またはCLOSEを入れる。ENDは、OPENしたすべてのファイルをCLOSEする働きがある。CLOSE命令は、複数ファイルをOPENしていたときに、「CLOSE♯n」の形で、特定のファイルだけを閉じることができる。
CTRL+STOPで中断したときも、CLOSEなどをダイレクトに実行してファイルを閉じておいたほうが無難。

ごめんなさい

付録ディスクにはＢＡＳＩＣピクニック用のファイルとして、「ＭＡＫＥＱＵＩＺ．ＢＰ４」「ＥＸＥ－ＱＵＩＺ．ＢＰ４」を収録していますが、これにはバグがあります。31ページ、33ページの掲載リストのように修正して使用してください。【リスト１】行120〜行140、行210、行220に変更があり、行215が追加。【リスト２】行120〜140に変更あり（変更点はリスト１とまったくおなじ）。

リスト2の緑地部分　まえのページと同様、引き出している部分以外のファイナル関係の部分に緑色の地をしている。こうしてみると、ランダムファイルの操作は、ほとんどバッファ用文字変数の扱いでしめられる。
バッファ用文字変数は取扱厳重注意の変数だ。まえのページにもちょっと書いたが、うっかり、ふつうの変数とおなじ代入文を使ってしまうと、その時点でバッファ用文字変数でなくなり、その後にやりとりしたファイルの内容は正常でなくなってしまうのだ。バッファ用変数は、できるだけそれとわかりやすい変数名を使い、LSET、RSET以外では代入をしないように細心の注意をはらいたい。

256バイトのカード

31ページのリスト1を実行して、ひとしきり問題を書きこんでいくと、そのときセットしていたディスクのなかに「QUIZ.DAT」というファイルが作られる。このファイルが、3択クイズ用の出題ファイルだ。

ファイル「QUIZ. DAT」は、たとえば、図1の左側、ディスクと書かれたほうの図がだいたいのイメージを表している。いくつかの細かな区切りを、256バイト単位で繰り返すデータファイルだ。

しかし、「QUIZ. DAT」というデータファイルそのものには、このフォーマットの情報が記録されてはいないということに注意する必要がある。

だから、いったんリスト1のようなプログラムでできあがったデータファイルを、別のプログラムで使用する場合、おなじ形でファイルのOPENとバッファへの変数割り当てをしなくてはうまくいかない。リスト1とリスト2の行140〜160がまったくおなじなのは、そのためだ。リスト2で、リスト1とちがう構造のバッファを設定すると、文字データがずれて、まったく無意味なデータになってしまうだろう。

データファイル自体は、たんにデータの集積に過ぎないので、読み出すときもバッファの構造をそのファイルを作ったときの構造にあわせないとうまくいかない。だから、リスト1とリスト2の、行140〜160はまったくおなじにしてあるのだ。

もっとも、リスト1と2を1本のプログラムにしてしまえば、1回のOPEN文とFIELD文で、問題作成・更新も出題もできるし、バッファの形式の取り違えもない。

リスト2の出題用プログラムは、きわめて単純なプログラムだ。リスト1とおなじ形式でランダムファイルをOPENし、バッファを設定し、「GET#1,〜」。すると、M$(0)〜M$(5)に1問ぶんのデータが入ってくる。それを画面に表示したり、正解かどうかを判定するときに使う。

このGET一発で、バッファ用変数にまとめてデータが入ってくるのが、なんだかラクした気分で気持ちがいい。256バイト書きこめるカードのファイルから、てきとうなカードを1枚ずつ抜き取る感覚だ。

ランダムファイルは、名前からむずかしそうな印象があるが、じっさいに使ってみると、ほとんどPUTしたりGETしたりするだけなのだ。電源を落としても内容がなくならない特殊なメモリとして、いろいろな活用法が考えられるだろう。

■図1　ディスクのファイルとメモリのバッファのやりとり

■リスト2　3択クイズを出題する

EXE-QUIZ.BP4　＋α

```
100    'SAVE"EXE-QUIZ.BP4
110 CLEAR 1000:DEFINT A-Z:DEFUSR=342
120 COLOR 15,4,4:SCREEN 0:KEYOFF
130 CALL KANJI1:WIDTH 38
140 RESTORE 140:FOR I=0 TO 5:READ L(I)
:NEXT:DATA 112,34,34,34,1,36
150 OPEN "QUIZ.DAT" AS #1 LEN=256
160 FIELD #1,L(0) AS M$(0),L(1) AS M$(
1),L(2) AS M$(2),L(3) AS M$(3),L(4) AS
M$(4),L(5) AS M$(5)
170 M$="ブー！正解！":EK$=CHR$(27)+"K"
180 LR=LOF(1)/256
190 COLOR 15:CALL CLS:R=R+1
200 IF R>LR THEN 370
210 GET #1,R
220 PRINT "第";R;"問"
230 PRINT M$(0)
240 FOR I=1 TO 3:LOCATE 0,I*2+3
250   PRINT MID$("１２３",I*2-1,2)
   ;"：";M$(I)
260 NEXT
270 LOCATE 0,11:PRINT M$(5);CHR$(13);
280 IF STRIG(0)=0 THEN 280
290 A=USR(0):PRINT EK$;
300 INPUT "答えを番号で入力してくださ
い";A
310 IF A=VAL(M$(4))
    THEN SETBEEP 3,4:F=1
    ELSE SETBEEP 1,4:F=0
320 PRINT CHR$(30);EK$;
330 IF F=0 THEN COLOR 8
340 BEEP:PRINT MID$(M$,6*F+1,6);
350 TIME=0:FOR W=0 TO 100:W=TIME:NEXT
360 GOTO 190
370 CALL CLS:SETBEEP 1
380 PRINT "問題終了":END
```

ファイルのデータをバッファに読みこむ

GET＃n,〈レコード番号〉

そのデータファイルを作ったときのバッファの設定のしかたなどを、そのまま真似ておけば、GET命令を1回しただけで、バッファ用変数群(図2のバッファの図)にデータが一度に入ってくる。リスト2で重要なファイル関係の命令は、これだけだ。あとは、持ってきたデータの中身に応じて、表示したり、判定の材料に使ったりすればいいわけだ。ランダムファイルのGET文は、それまで空欄だったバッファ(じっさいはまえのデータが入っているのだが)に、GET文を1回実行するだけでどっとデータが流れこむ……そんなイメージがある(図2)。

■図2　GET一発でぜんぶ読みこむ

⬆画面の上のほうの出題を読んで、正解と思われる選択肢の番号を入力する。この画面でスペースキーを押すと「やさしい〜」というヒントメッセージが消え、「答えを〜」と答えをきいてくる。1〜3の数値を入力してリターンキーを押すとそれが正解か誤りかを判定してくれる。入力された数値と、M＄(4)の数値を比較しているのだ

リスト2の解説

【準備】付録ディスクから
EXE−QUIZ．BP4
というファイルを自分のディスクにコピーし、31ページ下の「ごめんなさい」に従って修正し、おなじファイル名でSAVEしなおす。そのディスクを書きこみ可にしてディスクドライブにセットしておく。

31ページのリスト1を使って、あらかじめ出題するクイズのデータファイル「QUIZ．DAT」をそのディスク上に作っておく。

【使い方】実行すると、「QUIZ．DAT」に入っている問題の数だけ出題する。問題がなくなると「問題終了」と表示してプログラムを終える。各出題は、問題番号、問題、選択肢の順で画面に表示される。画面のいちばん下に右詰めで表示されるメッセージは、その問題へのヒント。スペースキーを押すと、「答えを番号で入力してください」と表示されるので番号(1〜3)を入力してリターンキーを押す。間違ったら「ブー／」、正解なら「正解／」と表示してそれぞれにビープ音を出す。2秒弱待って、次の問題へ進む。リスト2では、最初から順に出題しているが、レコード番号の設定をくふうすれば、出題順をいろいろに変えることもできる。

【プログラムの補足】●行290のEK＄には、行70で設定しているエスケープシーケンス用の文字列が入っている。これは、カーソル以降の文字を消去するエスケープシーケンス。ヒントメッセージを表示したあと、答えを入力するときにその行をきれいにする必要があるためだ。●行310の変数Fは、判定メッセージを表示するときの、文字色を決めるためのもの。Fが0、つまり、答えが間違っていたら、そのときだけ文字色を赤くしているのだ。「ブー／」という字だけ赤く表示する。

2桁スクリーンのミステリーに挑戦

BASICテクニック

#35 YJKミステリーツアー

宝の持ち腐れということばがある。MSX2+ユーザーのうち何人が、SCREEN12という不思議なスクリーンモードを使ったことがあるだろうか。

2桁のスクリーンモード SCREEN10は、やや精度のおとる自然画モード上にSCREEN5なみのグラフィックを書きこめる(0～15のパレットコードが使用可能)。SCREEN11は、SCREEN10と同様の自然画モード+RGBグラフィック機能だが、0～255のカラーコードの形でLINE文などで直接VRAMを操作でき、データの調整で、YJK方式のデータにも、RGB方式のデータにもできる。SCREEN12は、すべてYJK方式。背景色や前景色をうまく選べばリスト1のように文字を書いたりもできるが、絵柄の上に文字や図形を表示するのは至難の業。ちなみに日本のMXS2+には、SCREEN9がないが、これはハングル(韓国の文字)表示用のスクリーンモードだ。

YJK方式 パソコンなどはふつう、R、G、Bデータで色を再現する「RGB方式」をとっているが、テレビやビデオでは、色を輝度と色差で再現するので「色差方式」(色差とは3原色の値から輝度の値を引いたもの)と呼ばれている。MSX2+のYJK方式は、ビデオで使われているYUV方式にヒントを得て考え出された色差方式の一種。人間の目は色の違いよりも輝度の違いに敏感なんだそうで、このYJK方式では、輝度(Y)のデータは細かく、色相(JK)のデータは粗く持つようになっている(31～32ページ参照)。

「自然画」のその後

今回は、2+以降の話だ。

MSXというパソコンのそもそもの気持ちは「互換性」にあり、そのために、2+ユーザーでも、Mファンに投稿するプログラムはMSX2や1で動く範囲で作ってくれるアマチュアプログラマが多い。

しかし、現時点ではユーザーの数からいって、MSXの標準機はMSX2+になってしまっている。Mファンの読者調査によれば、約70パーセント強がMSX2+ユーザーなのだ。

†

MSX2+の「+」の部分は、漢字BASICや漢字ROM、FM音源(厳密にはオプションだが)、縦横スクロール機能、そして自然画モードだった。

自然画モード……。なんて、懐かしいことばだろう。2+新発売当時、発表されるソフトは、ことごとく「自然画モードに対応」していて、おまけの自然画が入っていたものだが、おまけの域を超えないまま、いつのまにか「自然画」の名はスペックから消えていった。ユーザーレベルでは自然画の取りこみなどはほぼ不可能なので、どうしようもなく、わたしたちの自然画モードは開店休業状態だった。

しかし、最近、AVフォーラムやCGコンテストで、この自然画モードを扱う投稿者が出てきた。今月はファンダムにも自然画モードを使った作品がある(38ページ『3D立体』)。

†

自然画モードとは、BASICでは2桁のスクリーンモード、つまり、SCREEN10、11、12のことだ。

とくに、純粋に自然画モードであるSCREEN12では、正味1万9268色もの色を同時表示できる。色数が豊富だと右ページの写真のようにこまやかなグラデーションが可能になる。

ところで、SCREEN12が使っているVRAMは、SCREEN8のVRAMとまったくおなじサイズだ。

SCREEN8は、横256ドット×縦212ドットの画面で、1ドットにつき1バイト(256とおりのものを区別できる)の色情報を持っていて、256色を同時表示できる仕組みだ。SCREEN12も横256ドット×縦212ドットの画面で、しかもおなじVRAMサイズなのに、なぜ、表示可能な色がこんなにも圧倒的にちがうのか。

SCREEN12は、ドットごとの色情報を、RGB方式ではなく、きわめてミステリアスなYJK方式で処理しているからだ。おなじVRAMを使って約75倍の色数を表現できるYJK方式とは？ 以下次ページ。

●リスト1　YJKの色見本

```
100 SCREEN 12:COLOR 255,0,0:CLS
110 DEFFNJK(N)=(N+32)MOD64
120 DIM A%(1026)
130 OPEN "GRP:" AS #1
140 FOR Y=0 TO 31
150   LINE (Y,0)-STEP(0,63),Y*8
160 NEXT
170 FOR K=0 TO 63
180   KL=FNJK(K) AND 7:KH=FNJK(K)¥8
190   FOR A=0 TO 7
200     PSET (A*4  ,K),KL,OR
210     PSET (A*4+1,K),KH,OR
220   NEXT
230 NEXT
240 COPY (0,0)-(31,63) TO A%
250 FOR J=0 TO 23
260   JV=J*2.74
270   JL=FNJK(JV) AND 7:JH=FNJK(JV)¥8
280   CX=(J MOD 8)*32
  :CY=(J ¥  8)*64
290   COPY A% TO (CX,CY)
300   FOR A=CX TO CX+31 STEP 4
310     LINE (A+2,CY)-STEP(0,63),JL,,OR
320     LINE (A+3,CY)-STEP(0,63),JH,,OR
330   NEXT
340 NEXT
350 DRAW "BM8,200":PRINT #1,"Y=0~31 K=-3
2~31 J=-32~30(step3)
360 GOTO 360
```

リスト1のプログラム解説

このプログラムは、YJK方式によるグラデーションのこまやかさや、色数の豊富さを見るためのもの。単純にVRAM上のYJKデータを操作しただけなので、実際にはおなじ色のドットが数多く含まれている。YJKとRGBの関係は次ページで解説する。

【プログラム解説】

●行100　画面初期設定。前景色を白、背景色と周辺色は黒にしておく。●行120　J、K(色相)は6ビットの符号付きデータ(－32～31)で、最上位ビットは＋－の符号を表す。BASIC上は0～63の数値を扱うので、じっさいには32以降がマイナスのデータになり、0から順に画面に書きこむとグラデーションの上半分と下半分が逆になる。FNJK(n)はそれを正常化するためのもの。●行120　配列A%は途中でYとKの全組み合わせブロックを保存するためのもの。●行130　最下行のメッセージ用にグラフィック画面をオープン。●行140～160　縦1ラインごとにY(輝度)のデータを0～31に変化させて64ドットのラインを描画。横32ドット×縦64ドットの大きさで、なめらかな無彩色のグラデーションができる。●行170～230　そのブロックの上から、横1ラインごとにK(R系色相)データを－32～31に変化させて書きこむ。Kデータは33ページの図のように書きこむべき場所が決まっているので、PSETと、ロジカルオペレーションのORを使って、該当するところだけにデータを書きこんでいる。KL=Kデータの下位3ビット、KH=Kの上位3ビット。●行240　以上の結果描画されたYとKのグラデーションブロックをいったん配列A%に入れる。●行250～340　配列A%を横8個、縦3個に並べ、ブロックごとのJデータを24段階に変化させて書きこむ。JVは0～23に変化するループ用変数Jを0～63のJデータに変換したもの。JL=Jデータの下位3ビット、JK=上位3ビット。CX、CY=各ブロックの表示開始位置(行280)。行300～330が各ブロックに同一のJデータを書きこんでいるところ。●行350　文字表示。

Y=0~31 K=-32~31 J=-32~30(step3)

↑リスト1の実行画面。1ブロックがYとKのすべての組み合わせを含み(1ドット単位)、Jはブロックごとにほぼステップ3で変化する。これですべての組み合わせの8分の3

Y=0~31 J=-32~31 K=-32~30(step3)

↑リスト1のJとKの立場を入れ換えて表示してみるとまたちょっと感じが変わる。いずれにしろこのこまやかなグラデーションはRGB方式では絶対に出せない

なんとなくYJKが見えてきた

なぜ、VRAMの大きさがおなじなのに、SCREEN8は256色で、SCREEN12は1万9268色なのか。

そんなことはとてもありそうに思えないが、YJK方式は、じつに巧妙な仕組みでこれを実現している。

■RGBからYJKへ

そもそも、YJKデータとはなにかというと、図1の右の式によってRGBデータを変換したものなのだ。逆に、あるYJKデータは、図1の左の式によってRGBデータに変換されて表示される。

このようにして得られるYJKデータのうち、Yは輝度、つまり明るさを表す。J、Kは色相、つまり色合いを表す。人間の目は、明るさの違いには敏感だが、色合いの違いには比較的鈍感なのだそうだ。RGBは、3原色のそれぞれの明るさで表すので、どの要素も手を抜くことができないが、YJKデータでは、色合いのデータ(JK)の部分で手を抜くことができる。

■JKデータは4ドットで共有

SCREEN12の画面に対応するVRAM(パターン名称テーブル)では、YJKデータが図2のように収まっている。

VRAMでは、この4ドット1グループあたりに4バイトのメモリを割り当てている。この1グループ内のドットを左からドット1～4と呼ぶことにする。

ドット1～4は、それぞれにY1～Y4という専用の輝度情報を持っている。各5ビットだから、おなじ色調で、32段階のグラデーションが表現できることになる。

輝度情報は、このように各ドットごとに別々になっているが、色相情報(JK)は、各ドット共通で、ドット1～4はどれもまったくおなじJKデータになる。J、Kはそれぞれ6ビットのデータで、上位下位3ビットずつが図2のように置かれている。したがって、ドット1～4の色相情報は、すべて

J=JL+JH×8

K=KL+KH×8

(ただし、J、Kどちらも32～63のデータは－32～－1を表す)で計算される。

■1ドットあたり17ビット!

すると、たとえば、ドット2の色情報の量は、まず輝度情報Y2、これは5ビットだ。次に色相情報JKが6ビットずつ。すると、ドット2の色情報は、あわせて17ビット!

これは、ドット1、3、4についてもまったくおなじで、各ドットごとに17ビットの情報が与えられることになる。

17ビットの情報量というと、2^{17}=13万1072とおりのものが区別できる情報量で、すると、色も13万1072色を表現できそうなものだが、じっさいには、YJKの組み合わせのうちには無意味なものや色の重複したものも含まれているので正味1万9268色ということになる。

YJK方式は、この色数と引換えに、4ドット単位でおなじ色相データを持つというハンデを引き受けているのだ。

YJKを扱う上で重要なのはまさにこの点で、自然画モードのスクリーンでは、つねに4ドット単位の大きな格子があると考えていないとうまくいかない。SCREEN12でなにも考えずにグラフィック命令を使用すると、SCREEN2の色化けよりもひどい状態になる。

■YJK/RGB混在モード

YJK方式のデータとRGB方式のデータが混在するSCREEN10、11では、このうちビット3にアトリビュートビットというのが追加され(図2)、そのぶんYデータは1ビット減っている。

このアトリビュートビット(A1～A4)が0のドットは、YJK方式によるドットとして扱われるが、アトリビュートビットが1になっていると、そのドットのYデータ(4ビット)をパレットコードとするRGB方式のドットとして扱われる(JKデータは無視される)。

SCREEN10～12の実際の使い方は……以下次号。

RGB　Red、Green、Blueの略で光の3原色の明るさの割合で色を表現する。どの要素も手を抜くことができないと、本文では書いたが、じっさいにはSCREEN 8でB(青)の要素が手を抜かれている。SCREEN 8は、1ドットあたりの色情報が1バイト(8ビット)なので、R、G、Bそれぞれにビットを割り当てようとしても割り切れない。そこで、なかでも人間の目がもっとも鈍感といわれるBだけ2ビット(つまり4段階)になっている。

パターン名称テーブル　VRAMで、おもに画面と対応したデータを持つところ。テキストモードでは、画面上の1マスに1バイトが対応していて、そのマスにある文字のキャラクタコードを持っている。SCREEN12では、画面の4ドットに対応して4バイトのYJKデータを持ち、図2のような構造になっている。ちなみにSCREEN 8では、画面の1ドットにパターン名称テーブルの1バイトが対応し、そのドットのカラーコード(0～255：1バイトの情報)を持っている。

JKデータ　図1の変換式で使うJ、Kは、どちらも－32～31の範囲の整数になるが、LINE文などでVRAMにJKデータを書きこむときは、マイナスの数値のときに困ってしまう。じつは、J、Kは、6ビットの符号付きデータで、最上位ビットが0のとき正、1のとき負を表す。しかし、LINE文などで扱うときは、その6ビットをそのまま通常の2進数として換算した値になる。たとえば、VRAM上の2進表現では、

JL……ØØØ(2進数)

JH……1ØØ

になっている。これを本文中のJを求める式に適用すると、

J＝0＋4×8＝32

しかし、この数値は、図1の変換式では使えない。JHの最上位ビットが、1なのでこのJはマイナスなのだ。同様に、VRAMとのやりとりで使うデータは、32～63という値をそのまま扱わないとうまくいかないが、RGB－YJKの変換式に使うときは、これを－32～－1にしなければならない。

■図1　YJKデータとRGBデータの変換式

YJK→RGB

$$R=Y+J$$

$$G=Y+K$$

$$B=\frac{5}{4}Y-\frac{1}{2}J-\frac{1}{4}K$$

RGB→YJK

$$Y=\frac{1}{2}B+\frac{1}{4}R+\frac{1}{8}G$$

$$J=R-Y$$

$$K=G-Y$$

※ただし、0≦Y≦31　－32≦J、K≦31
0≦R、G、B≦31　この範囲を超えた数値は超えたぶんだけ無視される(例：Rが40になった場合は、31になる)。また、式の結果は小数点以下第1位で四捨五入すること。

■図2　YJKデータとVRAMの関係その1（SCREEN12の場合）

	5ビット	3ビット
1ドット目	Y_1	KL
2ドット目	Y_2	KH
3ドット目	Y_3	JL
4ドット目	Y_4	JH

※4ドット1グループでJ、Kを共有

たとえば、2ドット目のYJKデータは
$Y=Y_2$
J=JH×8+JL
K=KH×8+KL

■図3　YJKデータとVRAMの関係その1（SCREEN10、11の場合）

	4ビット	1ビット	3ビット
1ドット目	Y_1	A_1	KL
2ドット目	Y_2	A_2	KH
3ドット目	Y_3	A_3	JH
4ドット目	Y_4	A_4	JL

※4ドット1グループでJ、Kを共有

たとえば、2ドット目の色データは
A_2がØのときYJKモード
$Y=Y_2$
J=JH×8+JL
K=KH×8+KL

A_2が1のときRGBモード
パレットコード$=Y_2$

闇に浮かぶモノトーン

BASICテクニック

#36 YJKでグラフィック遊び

SCREEN12で絵をかこうとすると、YJKマジックにはばまれ、色化けに悩まされ、やがて眠る。これではいけない。月にかわって、お仕置きよ！

アトリビュートビット　SCREEN12のYJK方式では、1ドットごとに輝度情報(Y)が5ビット(0～31)割り当てられているが、SCREEN10と11では、それを4ビット(0～15)に下げ、余った1ビット(ビット3：2進数にして下位から4桁目)をYJK、RGBの切り換え用に使っている。このビットが、0なら、YJK方式、1ならRGB方式になる。このビット3のことをアトリビュートビットという。アトリビュート(attribute)は、「属性」という意味で、よりひらたくいえば種類とか性質といったようなもののことだ。
そのドットに対応する1バイトの色データをAとすると、
A　AND　2＾3
が0なら、そのドットはYJKのドット、8(＝2＾3)ならRGBのドットとして表示される。
ちなみに、SCREEN12でグラフィックを表示している状態で、SCREEN10やSCREEN11を実行すると、自動的にすべてのドットのアトリビュートビットを1にしてくれる。

モノトーングラフィックの理由

前号に引き続き、MSX2＋以降の自然画モード、YJK方式について、なにかとアレしてみた。なんだかあいまいな言い方だが、なにしろナニがソレだからしかたがない。

SCREEN12でできること

SCREEN12では、正味1万9268色の豊富でこまやかな色が表示できるのだが、ただし、12ビットの色相情報が4ドットにまとめて1つしかないため、1ドット単位で自由に色を操作するわけにはいかない。たった1ドットのデータを変えただけで、そのドットを含む4ドットグループ全体の色相が変わって、いわゆる「色化け」を起こしやすいのだ。

だから、形のはっきりしたグラフィックをかくには、SCREEN12は適していない、という結論がいちおう出てくる。

しかし、輝度情報(Y：5ビット)だけは1ドット単位で持てるので、ある範囲のJKデータを均質化して、輝度情報だけを変化させれば、自由にグラフィックがかける。RGBでは望めなかった最高32段階のグラデーションを使用したグラフィックがかけるのだ。リスト1はそのサンプルだが、このように複数の色相を使ったグラデーションも可能だ。

SCREEN10と11について

YJKとRGBが混在しているSCREEN10と11は、描画するときの命令の使い方に違いがあるだけで、表示能力はまったくおなじだ。したがって、この2つのスクリーンモードのデータは完全な互換性を持っている。おなじ画像のデータなら、どちらのモードで見てもおなじように見えるというわけだ。

では、SCREEN10、11の兄弟と、孤高のSCREEN12とのあいだはどうか。

ぜひ、実験してみてほしい。SCREEN12でかいたグラフィック(BSAVEしたもの)を、SCREEN10、11でそのまま読みこんでみよう。たいていの場合、どぎつい画像になって現れるだろう。アトリビュートビットが1だったドットがRGB方式に変わるからだ。

このかぎりでは、SCREEN10、11とSCREEN12のあいだには、データの互換性がないように思えるが、じつは、コンバータともいえる機能がはじめから付いているのだ。

それを証明するのが右ページのリスト1だ。このプログラムは、グラフィックをかいたあと、SCREEN12からSCREEN10に移行しているのだ。

移行すると画面はどうなるか。不思議だがほとんど変わらない(プログラム解説参照)。

↑てきとうに選んだJKデータを4ブロックに分けて均質化し、Yデータだけで作ったグラデーションを乗せてみたのが、この画面

●リスト1 4つのグラデーション

```
100 SCREEN 12,0:COLOR ,0,0:CLS
110 R=ATN(1)/3:DIM A%(4597)
120 FOR I=0 TO 63:C=(I¥2)*8
130   CIRCLE (50,100-I),48-I¥4,C,R,R*11
140 NEXT
150 COPY (0,0)-(100,90) TO A%:CLS
160 FOR CX=0 TO 1:FOR CY=0 TO 1
170   X=CX*128:Y=CY*106
180   READ J,K
190   A$=RIGHT$("00000"+BIN$(J),6)+RIGHT
$("00000"+BIN$(K),6)
200   COPY A% TO (X+14,Y+6),,OR
210   FOR N=0 TO 31:FOR M=0 TO 3
220     LINE (X+N*4+M,Y)-STEP(0,105),VAL
("&B"+MID$(A$,10-M*3,3)),,OR
230   NEXT:NEXT
240 NEXT:NEXT
250 IF STRIG(0) THEN 280 ELSE 250
260 TIME=0:FOR W=0 TO 90:W=TIME:NEXT
270 '●●● SCREEN 12 <--> SCREEN 10
280 S=2-S:SCREEN 12-S
290 IF S THEN 260 ELSE COPY A% TO (14,6)
,,OR:GOTO 260
300 DATA  -2,-27,  3,  3, -7,  4, 10,-11
```

プログラム解説

リスト1の使い方　実行すると、まず白いグラデーションユニットをかき、いったん消えてから、上の写真のような画面に落ち着く。この画面をよく見てから、スペースキーを押すと、画面のモードがSCREEN10とSCREEN12とを往復するようになる。左上すみのブロックをよく見てほしい。うまくいっていればそこが脈動しているはずだ。

解説　●100　画面初期設定　●110　グラデーションパターン描画用の変数設定／パターン保存用の配列宣言　●120〜140　グラデーションパターンを描く【R：CIRCLE文の開始角度／C：描画するときのカラーコードに相当するデータ。このデータは、ビット0〜2を0にして、輝度(Y)だけでグラデーション効果を出している】　●150　グラデーションパターンを配列A%に保存する／画面消去　●160〜240　画面を4つのブロックに分け、それぞれにA%をセットし、その上に行180で読みこんだJKデータ(行300)を均一に書きこんでいく(行210〜230)　【CX、CY：ブロック単位の位置を表す／X、Y：各ブロックの左上座標／A$：JKデータ一時保存用／N、M：行210〜230のループ用。NはYJK方式のドットを4ドット単位で管理し、Mは各単位のなかでJKデータを正しい順に書きこんでいくためのもの　●250〜260　トリガー入力待ち／1.5秒待つ　●280　スクリーンモード切り換え【S：スクリーン切り換え用。　前者のときはSCREEN10、後者のときはSCREEN12になっている】　●290　SCREEN10なら行260へ、12ならもういちどA%を左上ブロックにORで上書きする。SCREEN10になった段階でグラデーションは細かさが半分になる。ふたたびSCREEN12に移っても粗くなったままだ。そこへA%をコピーするとふたたびグラデーションはもとのこまやかさにもどる　●300　4組のJKデータ

リスト2の行80について　Yの変換式の最初に、
Y=(B+R/2+G/4+1)¥2
とあるが、この+1は四捨五入するためのものだ。つまり、0.5を加えて小数点以下を切り捨てているのだ。この方法はよく見られる。

JKの組み合わせを探す

YJKとRGBの混合

SCREEN12からSCREEN10、11に行くときはアトリビュートビットがすべて1になるという目立たないが劇的な変化をとげる。しかし、SCREEN10と11は、たがいにまったくVRAMのデータを損なわずに行き来できる。

そこで、YJKとRGBを混合して使いたいときは、YJKグラフィックはSCREEN12でかいておき、SCREEN10や11に切り換えてから、RGBのグラフィックを上書きしたりすればいいだろう。単純に、16色のパレットを使ってガンガン描画したいのなら、SCREEN10、YJKのデータもいじる必要があるならSCREEN11にすればいい。

ただ、グラフィック画面への文字表示に関しては、SCREEN10ではなぜかRGBのデータとして扱われない。文字表示はSCREEN11で、アトリビュートビットのことを計算にいれた色データにすればうまくいくようだ。

RGBからYJKを計算

さて、いろいろな色を使ってYJKグラフィックをかくのはいまのところなかなかむずかしいが、適切なJKの値を探るための道具として、RGBからYJKデータを計算してみよう。

RGB方式のパレットでは、RGBのそれぞれについて、3ビット(0〜7)の段階しかなかったが、SCREEN12は、YJK方式によって、RGBそれぞれについて5ビットずつ(0〜31)の情報量を持てるようになった(RGBとYJKの関係については下図参照)。SCREEN10と11は、もう少し粗いはずだが。

YJKデータそのままでは、じっさいにどんな色になるか想像しにくいが、RGBの割合であれば、これまでの3ビットの経験からあるていど予測できるだろう。そこで、指定したRGBのデータから、その色を表示するためのYJKデータを教えてくれるプログラムを作ってみた

◉リスト2 RGB-YJK変換

```
100 SCREEN 0:COLOR 15,4,7:WIDTH 40:KEYOF
F:DEFINT A-Z
110 IF INKEY$>"" THEN 110
120 PRINT USING"R:## >>>";R;
130 INPUT R:IF R>31 OR R<0 THEN 120
140 PRINT USING"G:## >>>";G;
150 INPUT G:IF G>31 OR G<0 THEN 140
160 PRINT USING"B:## >>>";B;
170 INPUT B:IF B>31 OR B<0 THEN 160
180 Y=(B+R/2+G/4+1)¥2:J=R-Y:K=G-Y
190 PRINT
200 PRINT USING"R=### G=### B=###";R;G;B
210 PRINT USING"Y=### J=### K=###";Y;J;K
220 A$=RIGHT$("00000"+BIN$(J),6)+RIGHT$(
"00000"+BIN$(K),6)
230 FOR I=0 TO 3
240 PRINT USING"&&=&   & ";MID$("KLKHJLJ
H",I*2+1,2);"&B"+MID$(A$,10-I*3,3);
250 NEXT
260 IF STRIG(0) THEN 280 ELSE 260
270 '●●● YJK DISPLAY
280 SCREEN 12,0:COLOR ,0,0:CLS
290 FOR N=0 TO 7:FOR M=0 TO 3
300 LINE (N*4+M+100,80)-STEP(0,31),Y*8+V
AL("&B"+MID$(A$,10-M*3,3)),,OR
310 NEXT:NEXT
320 BEEP:BEEP:BEEP
330 IF STRIG(0) THEN 100 ELSE 330
```

```
R:31 >>>?
G: 0 >>>? 5
B:16 >>>? 24

R= 31 G=  5 B= 24
Y= 20 J= 11 K=-15
KL=&B001 KH=&B110 JL=&B011 JH=&B001
```

⬅前回入力値を参考にしながらRGBの値を変更していくと色の予測をつけやすい。KL〜JHも知っていて損はないと思う

◉1ドット目のRGBの計算例

R=Y0+J
G=Y0+K
$B=\frac{5}{4}Y-\frac{1}{2}J-\frac{1}{4}K$
Y0=&B10100
=20
J=JH×8+JL
=1×8+3=11
K=KH×8+KL
=(−2)×8+1=−15
∴R=31　G=5　B=24

※KH、JHは最上位ビットが1の数値は負として扱います。

◉図 4ドット単位のYJKデータの構造

(リスト2)。

R、G、B(各0~31)を入力していけば、その色のドットを表示するために必要なVRAMデータを表示し、そのあとスペースキーを押すと、実際にその色を表示し、スペースキーでもとにもどる。データ入力のときただリターンキーを押すと前回の入力値(これも表示)のままにしておける。

JKを探して

いろいろなJKの組み合わせとグラデーションを組み合わせてみたいときは、リスト3を利用してほしい。スペースキーを押すたびにランダムなJKの組み合わせをグラデーションで12種類ずつ見せる。31ページのリスト1のJKデータも、リスト3を実行しながら探した。

各ブロックの下に表示されている数字は、左がJ、右がKの値だ。とちゅう「これは」と思うものに出会ったら、CTRL+STOPして、F1キーを押せば、ディスクにそのときの画面をセーブすることができる。ファイル名はそのつど入力。

リスト4は、J、Kデータを入力して、そのデータでできるグラデーションを3種類デモしているだけのものだが、ここでは、行280に注目しておいてほしい。この行は、JKデータを変化させずに、Yデータだけを0にしているのだ。このあと、適当なYデータをORで書きこめば失敗しないわけだ。そうしないと、ところどころで妙に白い巣ができるだろう。行280のある場合とない場合とを見比べてみればわかるはずだ。

YJKでは、ロジカルオペレーションがひときわ重要だ。ロジカルオペレーションを軸にしたテクニックを研究する必要があるだろう。以下次号。

10 -9 / 14 27 / -3 5 / -17 18
9 22 / -23 3 / 4 -11 / -13 20
-9 -6 / -21 17 / -16 0 / 18 -7
MORE? (HIT SPACE KEY)

⬅一つのグラデーションをランダムなJKで

⬅斜めのグラデーションに注目

◉リスト3 ランダムグラデーション

```
100 SCREEN 12,0:COLOR ,0,0:SETPAGE 1,1:S
CREEN,0:CLS:KEY1,"GOTO 370"+CHR$(13)
110 OPEN "GRP:" AS #1
120 DEFINT A-Z:DIM R(2050)
130 FOR Y=31 TO 0 STEP -1
140   C=Y*8
150   LINE (31-Y,31-Y)-STEP((Y+1)*2-1,(Y
+1)*2-1),C,BF,PSET
160 NEXT
170 COPY (0,0)-(63,63) TO R:CLS
180 FOR CX=0 TO 3:X=CX*64
190  FOR CY=0 TO 2:Y=CY*64+CY
200   COPY R TO (X,Y),,PSET
210   J=INT(RND(1)*64)-32
220   K=INT(RND(1)*64)-32
230   A$=RIGHT$("00000"+BIN$(J),6)+RIGHT
$("00000"+BIN$(K),6)
240   FOR N=0 TO 15:FOR M=0 TO 3
250     LINE (X+N*4+M,Y)-STEP(0,63),VAL(
"&B"+MID$(A$,10-M*3,3)),,OR
260   NEXT:NEXT
270   COLOR 7:PSET(X+2,Y+54),0,TAND:PRIN
T#1,USING"### ###";J,K
280  NEXT
290  LINE (X,0)-(X,Y+63),7,,AND
300 NEXT
310 PSET(50,200),0,TPSET:COLOR 255:PRINT
 #1,"MORE? (HIT SPACE KEY)"
320 IF STRIG(0) THEN 180 ELSE 320
330 '●●● SAVING SCREEN
340 SCREEN 0:COLOR 15,0:INPUT"FILE NAME"
;F$:SCREEN 12:SETPAGE 1,1:BSAVE LEFT$(F$
,8)+".SCQ",0,&HD3FF,S
```

◉リスト4 JK指定のグラデーション

```
100 SCREEN 0:WIDTH 40:COLOR 15,4,7:KEYOF
F:DEFINT A-Z
110 IF INKEY$>"" THEN 110
120 INPUT "J,K";J,K
130 IF J<-32 OR J>31 OR K<-32 OR K>31 TH
EN 120
140 A$=RIGHT$("00000"+BIN$(J),6)+RIGHT$(
"00000"+BIN$(K),6)
150 SCREEN 12,0:COLOR ,0,0:CLS
160 FOR N=0 TO 63
170   FOR M=0 TO 3
180     LINE (N*4+M,0)-STEP(0,211),VAL("
&B"+MID$(A$,10-M*3,3)),,OR
190   NEXT
200 NEXT
210 FOR X=0 TO 127
220   LINE (X,0)-STEP(0,105),(X¥4)*8,,OR
230 NEXT
240 FOR X=128 TO 255
250   LINE (X,0)-STEP(255-X,255-X),((X-1
28)¥4)*8,,OR
260 NEXT
270 FOR X=255 TO 0 STEP -1
280   LINE (X,107)-(255,211),7,,AND
290   LINE (X,107)-(255,211),(X¥8)*8,,OR
300 NEXT
310 BEEP:BEEP:BEEP
320 IF STRIG(0) THEN 100 ELSE 320
```

ビット単位の配慮

BASICテクニック

#37 YJKに必要な技法

もう一度だけYJKの世界をさまよってみたい。手に入れたわずかな知識をもとに、色化けと多段階グラデーションの境目をふらふらと散歩してみよう。

グラフサウルスVer.2.0 SCREEN12モードのエディタは、YJKの性質にあわせてルーペのウインドウは横4ドット単位で動くようになっているし、複写も横4ドット単位でおこなわれるので、色化けは最小限にとどめられる。ただし、YJKのメーターでYは0～31なのはいいが、J、Kが0～63になっているので注意が必要だ。この場合、0～31はそのままの数値だが、32は－32を意味している。以降、33は－31、34は－30……63は－1のことだ。この変換をしておかないと、YJKとRGBの変換公式が成り立たないので注意。また、色相の変化の具合も、当然、－32から0を経て31にいたるまで連続的に変化していくのだということに注意。現在のメーターのままだと31と32のあいだで飛躍的に色相が変わってしまい調節しづらいのではないか。

アトリビュートビット SCREEN10と11の各ドットごとのデータ(1バイト)のビット3、つまり下位から数えて4番目のビット(2^3)は、そのドットがRGB方式か、YJK方式かを切り換える働きがある。このビットが1だと、RGB方式(そのドットのデータの上位4ビットをパレットコードとして表示)、0だとYJK方式(SCREEN12の場合と同様。ただし、Yデータはかならず偶数になる)で表示される。

YJKの上にRGBを乗せる

ビッツーの「グラフサウルスVer.2.0」には、MSX2＋以降用にSCREEN12モードがある。ここでは、フルYJKのグラフィック画面上で、YJKメーターを見ながら絵をかくことができる。ただし、ほかのモードのようにスイスイと絵がかけるわけではない。このモードには色化けという強敵がいて、そのために、ツール自身にペイント機能がなかったり、ペンの太さを変えられなかったりという制約がある。この制約は、技術的に難しいというよりも、ほとんど意味がない、という先天的な不自由さにも原因がある。なぜなら、SCREEN12のほとんどの色は、4ドット単位のまとまりでしか意味を持たないからだ。

だから、SCREEN12をはじめとする自然画モードは、CGをかく場としては「使えない」という風評が一時期立っていた。第一、ビッツー自身がSCREEN12のCGをほとんどかいていないところからも、その「使えなさ」は明らかなように見えた。

しかし、たとえば、5月号のCGコンテスト102ページを見よ。左下にさりげなく掲載されていたエレファンツエッグランクなCGは、SCREEN12の作品なのである。といわれてもわからない人は、右ページ右上の画面写真を見よ。これ以前にもSCREEN12の作品はあったが、グラデーションが不自然に目立つ、いかにもYJKな作品ばかりだった。しかし、この北海道・松宮英(あきら)(17歳)の作品『夜のえきにて……』は、柔らかく繊細な肌と影と髪の毛、紗(しゃ)をかけたようなほのかな輪郭線が印象に残る、ごく自然なCG作品に仕上がっている。この作品こそ、SCREEN12のCGの出発点になるのではないか。……なんだか、話がそれてしまったでおじゃる。

それはさておき、とにかくSCREEN12は色化けが手強くて、絵がかきにくい。そこで、道は2つある。

1つは、前回にも書いたように、一定の領域のJKデータを均質化してモノトーンでかくこと。『夜のえきにて……』もまさにそれで、肌、髪、影の部分のJKデータはすべておなじだ。

もう1つは、YJKでできる部分はYJKでやって、そうでない部分はRGBでやってしまうことだ。つまり、YJK、RGB混在のSCREEN10、11をうまく使うということ。このモードは、YJKの上にRGBを乗せる機能があるのだ。

右ページの6枚の写真で、特性を実感してほしい。

◈SCREEN12(フルYJK)

↑リスト２(33ページ)の最初の画面。ＲＧＢのうち、ＲとＧを１刻みで32段階変化させて作ったグラデーション(32ページ・リスト１)に輝度(Ｙ)248の文字を乗せた

↑ＳＣＲＥＥＮ12の「夜のえきにて……」。グラフサウルスで読みこみ、耳たぶあたりをルーペで拡大したでおじゃる。色相(ＪＫ)はぜんぶおなじ、Ｙデータだけ違うまろ

◈SCREEN10、11で同じデータを読みこむと

↑リスト１のグラデーションのデータ(ＳＣＲＥＥＮ12用)をＳＣＲＥＥＮ10で読みこんだときの画像。アトリビュートビット１のドットだけＳＣＲＥＥＮ５並みになる

↑ＳＣＲＥＥＮ12のデータをＳＣＲＥＥＮ10、11でじかに読みこむとこうなるぞよ。これは特殊な効果に使えるかもしれないまろが、予測がつかないでおじゃる

◈SCREEN10、11(YJK+RGB)

↑リスト２の２番目の画面。おなじデータでもある手順で読みこめばこのようになじむ。その上にＲＧＢで白い文字を乗せ、直線と円をかき、円内を塗りつぶしてみた

↑正しい手順で読みこめばＳＣＲＥＥＮ10でもきれいな「夜えき」ができるまろ。よく見ると影の変化が粗くなってちょっとおばさんみたいな感じでござりまする。信長じゃ

4ドットの公式、ビット3の法則

SCREEN10とSCREEN11　SCREEN10とSCREEN11の違いは、画面になにかを書きこむときだけ出てくる。①SCREEN10で使えるカラーコードは0～15(SCREEN5なみ)だが、SCREEN11で使えるカラーコードは、0～255（SCREEN8）なみ②SCREEN10で使用したグラフィック命令は、各ドットごとのデータの上位4ビットだけを変化させ、同時にビット3を1にする(RGB方式でしか書きこめない)が、SCREEN11で使用したグラフィック命令は全ビットを変化させ、ビット3に対する特例はない(自由に設定できる)

リスト2は、SCREEN12(フルYJK)とSCREEN10、11(RGB+YJK)のサンプルを交互に表示するプログラムだ。ここでたいせつなのは
①SCREEN12の状態でグラフィックを表示しているとき、SCREEN10を実行してもグラフィックは消えない
②しかも、そのとき、SCREEN12用のYJKデータをSCREEN10用のYJKデータに変換してくれる(ただし、グラデーションが粗くなる)
という2点だ。

これはSCREEN12とSCREEN11の場合でも、まったくおなじだ。

†

SCREEN10、11を使う場合、とりあえず、各ドットのビット3(アトリビュートビット)を意識しなくてはいけない。

前述の②はようするに、ビット3をすべて0にしてしまうという意味だ(前回はこれを逆に書いていた。おわびして訂正いたしまするのじゃ)。ビット3が1になっていると、SCREEN10、11では、上位4ビットをRGB方式のデータとして表示する。たとえば、前ページの中段においた画面はそういうドットをふんだんに含む例だ。

†

上の4つの正方形のブロックは、すべておなじデータだが、データを置く位置が違っていたためにこうなった。リスト3は、4ドットごとの区切りを無視してCOPYした場合の変化を見るためのプログラムで、これを実行すると、上の4つのパターンをくりかえしながらブロックが右へ進んでいく。

これはこれで1つの効果として使えないこともないが、通常は、4ドットの法則を守っていないとおちおち四角もかけない。X座標を次のように管理すれば、とりあえず4ドットの法則から逸脱せずにすむ。

(X¥4)＊4

このXに0～3を代入してみよう。ぜんぶ0になる。4～7は4、8～11は8というふうに4刻みにしか増えない。

●リスト1　RGBで計算したグラデーション

```
100 SCREEN 12:COLOR ,0,0:CLS:SETPAGE 1,1
:SCREEN ,0:CLS
110 GX=0:DY=1:GY=DY*32
120 FOR B=0 TO 31 STEP 4
130   FOR R=0 TO 31
140     FOR G=0 TO 31
150       Y=(B+R/2+G/4+1)¥2
160       J=R-Y
170       K=G-Y
180       A$=RIGHT$("00000"+BIN$(J),6)+R
IGHT$("00000"+BIN$(K),6)
190       FOR N=0 TO 3
200         PSET (N+GX*4,GY),Y*8+VAL("&B
"+MID$(A$,10-N*3,3))
210       NEXT
220       GY=GY+1
230     NEXT
240     GX=GX+1
250     IF GX>63 THEN GX=0:DY=DY+1
260     GY=DY*32
270   NEXT
280 NEXT
290 IF NOT STRIG(0) THEN 290
300 BSAVE "RGB-GRD.SRS",0,&HD3FF,S
```

プログラム解説

動作内容　SCREEN12のRGB各5ビットのうち、Bを8段階にわけ、それぞれについてR、Gを32段階ずつ変化させて、各ブロックについてグラデーションを作る。

プログラム解説　●行100　スクリーン12に／背景色、周辺色を0に設定／CLS／ページ1に切り換え／スプライト初期化(ページ1の場合、スプライトでゴミがついていることがあるので)／CLS　●行110　変数初期化【GX：ドットを置く4ドット単位ブロックの先頭X座標／DY：各グラデーションを開始するY座標／GY：ドットを置くY座標】　●行120　B(青)のループ開始(ステップ4)　●行130　R(赤)のループ開始　●行140　G(緑)のループ開始　●行150～170　RGBをYJKに変換【Y：輝度データ／J、K：色相データ】　●行180　JKデータを2進数12桁の文字列に変換(行200で描点時に使用)　●行190　Nのループ開始(4ドット単位ブロックのなかの第1ドットから第4ドットまでのループ)　●行200　指定されたYJKの色で4ドット置く　●行210　Nのループ閉じ　●行220　ドットを置く位置を1ドット下へ　●行230　Gのループ閉じ　●行240　ドットを置く4ドット単位ブロックを1ブロック右へ　●行250　右端まで来たらGXをもとにもどしてグラデーション開始位置を1(32ドットぶん)下げる　●行260　ドットを置く開始Y座標設定　●行270～280　Rのループ、Bのループ閉じ　●行290　トリガー入力待ち　●行300　グラフィックを「RGB－GRD.SR5」としてセーブ

⬆4つのブロックのデータそのものはまったくおなじ(なにしろ単純にコピーしているだけだから)なのだが、左肩のドットの位置がMOD4で0、1、2、3のところに置かれた場合のそれぞれの表示サンプルだ。左端がもともとの状態

⬆SCREEN12のまま、漢字メッセージを裏画面経由でXORコピーした。きれいに文字を乗せることができた

†

リスト4は、例として『夜のえきにて……』にタイトルを漢字で入れるためのものだ。このCGは、JKデータが均一になっているので、JKデータを0にしたグラフィックをORなどで重ねると文字が色化けしない。この場合は、背景色0の裏画面にYデータ248(&B11111000)の文字をかいておき、それを絵の上にXORでCOPYした。こうすると、うまくいけば、暗いところでは文字が明るく、明るいところでは文字が暗くなるからだ(へたすると見えない可能性もある)。

†

じつは、今回のメインテーマはYJKでごまかしながら塗りつぶす疑似ペイントの予定だったのだが、まったくうまくいかなかった。もし、計算だけで、色化けを最小限度に止めながら色を塗っていくルーチンを作っている人がいたら教えてほしい。

では、YJKシリーズを終わります。ぞよ。

●リスト2　SCREEN12と10、11の特徴

```
100 SCREEN 12:COLOR ,0,0:CLS:SETPAGE 1,1
:SCREEN ,0:CLS
110 OPEN "GRP:" AS #1:COLOR &B11111000
120 BLOAD "RGB-GRD.SRS",S:BEEP
130 COPY (0,32)-(255,159),1 TO (0,32),0
140 PSET (12,100),0,TOR:PRINT #1,"SCREEN
 12":GOTO 190
150 SETPAGE 0,0:SCREEN 10:LINE (0,32)-(1
27,85),8,,PSET:CIRCLE STEP(0,0),30,8:PAI
NT STEP(0,1),4,8
160 SCREEN 11:PSET (12,100),0,TOR:PRINT
#1,"SCREEN 10 OR 11"
170 A$=INPUT$(1) '●●● HIT ANY KEY
180 SCREEN 12:SETPAGE 1
190 A$=INPUT$(1) '●●● HIT ANY KEY
200 GOTO 150
```

プログラム解説

動作内容　リスト1で作成したデータを読みこみ、SCREEN10と11、およびスクリーン12でそれぞれの特性を示すように表示する
プログラム解説　●行100　画面初期設定　●行110　グラフィック画面へ文字を表示する準備／前景色のカラーコードとしてYデータのみフルになったコードを設定　●行120　グラフィックファイルの読みこみ　●行130　ページ1のグラデーションをページ0のおなじ位置にコピー　●行140　ページ1のグラデーションの上に「SCREEN 12」を表示　●行150　ページ0に切り換え／スクリーン10に／赤い線で直線をかき、円をかき、円の中を青く塗る　●行160　スクリーン11に／「SCREEN 10 OR 11」を表示　●行170　キー入力待ち　●行180　スクリーン12に／ページを1に切り換える　●行190　キー入力待ち　●行200　行150へもどってやりなおし

●リスト3　4ドットの法則

```
100 SCREEN 12:COLOR ,0,0:CLS:SETPAGE 1,1
:SCREEN ,0:CLS
110 BLOAD "RGB-GRD.SRS",S
120 SETPAGE 0,0
130 FOR X=0 TO 100
140 COPY (112,79)-STEP(31,31),1 TO (X+32
,70),0,PSET
150 TIME=0:FOR W=0 TO 60:W=TIME:NEXT
160 CLS
170 NEXT
180 GOTO 130
```

プログラム解説

動作内容　リスト1で作ったグラデーションから一部を抜き出し、それを1ドット単位で動かしてみる
プログラム解説　●行100　画面初期設定　●行110　グラフィックファイルの読みこみ　●行120　ページ0に切り換え　●行130　Xのループ開始(ここでSTEP4を加えると同一のグラフィックが横にスーッと動くようになる)　●行140　グラデーションの一部を(32, 70)から(132, 70)まで1ドットずつ次々にコピーしていく　●行150　1秒待つ　●行160　画面消去(コピーされたグラフィックがつながらないように)　●行170　Xのループ閉じ　●行180　行130に飛ぶ

●リスト4　漢字の重ね焼き

```
10 CALL KANJI
100 SCREEN 12:COLOR ,0,0:CLS:SETPAGE 1,1
:SCREEN ,0:CLS
110 BLOAD "YORUEKI.SRS",S
120 SETPAGE 0,0:SCREEN ,0:CLS
130 COLOR &B11111000:LOCATE 0,0:PRINT "
夜のえきにて……"
140 SETPAGE 1,1
150 COPY (0,0)-(16*8-1,15),0 TO (83,145)
,1,TXOR
160 GOTO 160
```

プログラム解説

動作内容　『夜のえきにて……』にタイトルを重ねる
プログラム解説　●行10　漢字モードに　●行100　画面初期設定　●行110　グラフィックファイル(ここでは『夜のえきにて……』)の読みこみ　●行120　ページ0に切り換え／スプライト初期化／CLS　●行130　前景色のカラーコードとしてYデータのみフルになったコードを設定／座標(0, 0)／「夜のえきにて……」を表示　●行140　ページ1に切り換え　●行150　ページ0に表示しておいたタイトル文字をページ1の(83, 145)にTXORで複写(コード0は複写せず、そうでない部分は複写先とXORの論理演算をして複写)　●行160　行160へ(永久ループ)

BASICピクニック 外伝 Q&A特集

VRAMの内部をピクニックするためにはじめたこのコーナーもそろそろ模様替えの年頃になった。次号からの心機一転を期して、今回はよく寄せられる代表的な質問をまとめて、誌面が許すかぎり答えてみた。

Q マシン語のセーブのしかたを教えてください。

（羽柴秀吉／愛知・13歳）

マシン語のセーブは、かんたんです。

たとえば、メモリの&HDØØØ～&HD3ØØ番地にマシン語プログラムが入っているとすると、このマシン語プログラムをセーブするには、

BSAVE"〈ファイル名〉",&HDØØØ,&HD3ØØ

でOKです。ここで、ファイル名に続く2つのパラメータが、マシン語プログラムの開始アドレス、終了アドレスになっているのは明らかでしょう。ほんとうはこのパラメータのあとに「実行開始アドレス」を指定することもできるのですが、ここでは省略します。

また、こうしてセーブしたマシン語ファイルは、たとえば以下のような命令でロードして実行できます。

BLOAD"〈ファイル名〉",R

アドレスを指定する必要もありません。このように、マシン語のセーブ、ロードはひじょうにかんたんです。

ただし、セーブするためのマシン語プログラムを組むのは、けっこうたいへんです。でも、「じゃあ、マシン語の組み方を教えてください」という質問に答えるには半年くらいかかります。

Q ディスクのなかでどんどんファイルが増えて困ってます。

（揚貴妃／茨城・18歳）

いらないファイルが増えて困っている、という悩みだと思いますが、その場合、2つのケースが考えられます。

①妖怪のしわざで知らないファイルがどんどん増殖していく

②セーブするばかりでファイルの削除をしないので増える一方

①の場合は、質問するところが違います。ゲゲゲの鬼太郎とか美少女仮面ポワトリンとかお坊さんとかにきいてください。

②の場合は、問題です。プログラムを作ったりセーブしたりできても、ファイル削除の方法を知らないなんて……。あるいは、ファイルを消すにはフォーマットしかないと思っていたのでしょうか。次の命令を使えば、不要なファイルだけをディスクから削除することができます。

KILL"〈不要なファイル名〉"

ワイルドカードも使えますが（46ページ参照）ヘタをすると必要なファイルまで消してしまいかねないので注意してください。

Q エスケープシーケンスがうまくできません。

（松平元康／静岡・43歳）

エスケープシーケンスは、とても楽しい機能ですが、①PRINT文のなかで使う、②テキストモード（SCREENØか1）で使う、という2つを守らないとうまくいきません。それから、大文字、小文字の区別も厳密にする必要があります（下の表はすべて大文字を使うものばかり。x、yは座標数値を入れます）。

また、ダイレクトモードで試しても効果がわかりにくいので、プログラムにしてみるといいと思います。

●代表的なエスケープシーケンス

エスケープシーケンスの書式	機　能
PRINT CHR$(27)+"Y"+CHR$(32+y)+CHR$(32+x)	カーソルを(x, y)の位置に移動 ※LOCATE x, yと同じ
PRINT CHR$(27)+"L"	カーソル位置に1行挿入して下へスクロール
PRINT CHR$(27)+"M"	カーソル位置に1行削除して上へスクロール
PRINT CHR$(27)+"K"	カーソル位置から右側の文字を消去
PRINT CHR$(27)+"J"	カーソル位置から画面の終わりまでを消去

Q SELECTキーやESCキーはBASICでは使えないんですか?

（司馬遷／島根・18歳）

使えます。使えますが、この2つのキーにはあらかじめ機能が与えられていないので、押してもなにも起こりません。ただ、

SELECTキー………CHR$(24)
ESCキー………………CHR$(27)

というふうに、それぞれコードが割り当てられているので、これらのキーを押すと割りあてられたコードがちゃんとMSXに送りこまれているのです。

ちなみに、この2つのキーを押してもほとんどなにも起こりませんが、違う点が1つだけあります。それはなんでしょう。答えは、この項のいちばん最後にあります。

ところで、この送りこまれるキャラクタコードを生かしてBASICプログラムに利用することができます。

たとえば、

A$=INKEY$

などで、押されたキーのキャラクタコードをA$に入れておき、

IF A$=CHR$(24)～

というふうに判定すれば、プログラム中でSELECTキーになんらかの機能を持たせることができるわけです。市販ソフトなどの場合でも、原理的にはまったくおなじことなのです。

さっきの答え。挿入モードのとき、ESCキーを押してもなにも起こらないが、SELECTキーを押すと挿入モードが解除される。

Q コントロールコードってなんのためにあるの？

（小早川秀秋／兵庫・17歳）

それは………………やっぱりコントロールするためではないでしょうか。

コントロールコードとは、０〜31、および127のキャラクタコードのことです。コントロールコードがMSXに送りこまれると、そのコードに割り当てられたさまざまな働きをします。

カーソルキー（28〜31）、RETURNキー（13）、INSキー（18）、DELキー（127）、BSキー（8）、CLS／HOMEキー（12／11）は、それぞれカッコのなかのコントロールコードを送りこむための特別なキーです。ESCキー（27）、SELECTキー（24）も特別な機能はありませんが、まったく同様です。

コントロールコードを入力するのには、あと２種類の方法があります。

１つは、CTRLキーを押しながら、別のキーを押す方法です。プログラミングに慣れている人はCTRL＋EやCTRL＋Nを有効に使っていることが多いようです（右表参照）。

もう１つの方法は、ファンクションキーに登録するやり方です。たとえば、F１キーにビープを鳴らす機能（CTRL＋G：コントロールコード７）を設定したいときは、

KEY1，CHR$(7)

を実行すればOK。これでF１キーを押すたびにピッピッと鳴ります。

●おもなコントロールコード

キー操作	コード	機　　能
CTRL＋B	2	カーソルを直前の区切りへ移動
CTRL＋C	3	入力待ち状態の終了
CTRL＋E	5	カーソル以下、行の終わりまでを消去
CTRL＋F	6	カーソルを次の区切りへ移動
CTRL＋G	7	ビープ音を鳴らす
CTRL＋H	8	BSキーと同じ
CTRL＋I	9	TABキーと同じ
CTRL＋J	10	行送り
CTRL＋K	11	HOMEキーと同じ
CTRL＋L	12	CLSキーと同じ
CTRL＋M	13	RETURNキーと同じ
CTRL＋N	14	カーソルを行末へ移動
CTRL＋R	18	INSキーと同じ
CTRL＋U	21	カーソルのある行の文字をすべて消去

Q ゲームを止めると字が変になってしまう。

（松永久秀／京都・13歳）

質問の状況がちょっとはっきりしませんが、おそらく「ファンダムのプログラムを走らせて、CTRL＋STOPで止めると、文字の形や色が変わってしまっている」という意味でしょう。

これは、SCREEN１で作られたゲームではよく見られ、同時にMSXの大きな特長の１つでもあります。

文字には、フォント（文字の形）情報と、色情報があります。このフォントと色を変えて、ゲームのキャラクタや背景に使ったりしています。

その方法は、原理的にはスプライトパターンとおなじなのですが、スプライトのように専用の命令はありません。

たいていはVPOKEを使ってVRAMに直接パターンデータを書きこんでいます。そうでない場合もマシン語で書きこんでいるだけです。

SCREEN1で、キャラクタコードAの文字の場合、フォントはVRAMのA×8番地から8バイト、スプライトパターンデータと同様に入っていて、色はVRAMの&H2000＋A¥8に前景色×16＋背景色の形式で入っています（ただし、8文字ごとのグループでしか色を指定できない）。この情報をもとにいろいろ試行錯誤すれば、わかる人はわかる、わからない人はわからないでしょう。

↑「A」、「a」、「あ」の文字の色をそれぞれに変えてみた例

↑上とおなじ文字を横方向に２倍にした例（いわゆる「太文字」）

↑これもやっぱり上とおなじ文字。色も形ももうひとひねりしてみました

Q スプライトに変な字を代入してることがあるけれど……。

（細川忠興／熊本・14歳）

えーと、それは

SPRITE$(0)＝"<~ロロ~<B♥"

みたいなやつでしょうか。

このやり方は、BASICのマニュアルにはきちんと書いてないことが多いのですが、ようするに、

SPRITE$(0)＝CHR$(60)＋CHR$(126)＋CHR$(219)＋CHR$(219)＋CHR$(126)＋CHR$(60)＋CHR$(66)＋CHR$(129)

とおなじなのです。というのも、たとえば、

PRINT CHR$(60)＋CHR$(126)＋CHR$(219)＋……

を実行すると、

<~ロロ~<B♥

と表示されます。つまり、このCHR$の列と変な文字列とは同等なのです。文字列方式を使うと、プログラムが短くてすみますが、キャラクタコード０〜31と127はコントロールコードなので使えないという制約もあります。

Q 「多色刷り」ってなに？

（織田信長／岐阜・16歳）

本来、カラー印刷のことですが、Mファン用語では、いわゆる疑似SCREEN2モードのことをいいます。

SCREEN1での文字の色は、8文字のグループごとに前景色、背景色の2色しか指定できません。たとえば、「A」を赤い地に白い文字に指定すると、「@」〜「G」もその指定になってしまうという不自由さがあります。

ところが、SCREEN1で

DEFUSR＝&H7E：A＝USR(0)

を実行すると、１文字に対して16色（１ラインごとに２色）を独立に指定することができるようになるのです。

これが多色刷りです。通常のSCREEN1では、VRAMのカラーテーブルはぜんぶでたったの32バイトですが、多色刷りモードでは2048バイトに増えます。それにともなって、各文字の色の設定のしかたや、特別な事情などがあるのですが、それを解説しているスペースはないので、右にサンプルプログラムをつけておきます。興味のある人は、これを打ちこみ、あれこれいじって探ってください。もしかしたら、BASICピクニックでまたやるかもしれません。

↑下のプログラムを実行して、プログラムリストを表示してみた画面

```
10 COLOR ,,0
20 SCREEN 0:WIDTH 80:WIDTH 40
30 SCREEN 1:WIDTH 32
40 DEFINT A-Z:DEFUSR=&H7E:A=USR(0)
50 FOR I=0 TO 6143
60 IF I MOD 8=0 THEN D=I¥8:VPOKE 6144+D,D MOD 256
70 VPOKE &H2000+I,(RND(1)*13+2)*16 AND &HF0
80 NEXT
90 CLS:LIST
```

『Simple ASM』再発売記念

本格マシン語講座連載開始予告と体験版『Simple ASM』

■Simple ASMとは

BASICプログラムを組むような感覚でアセンブリ言語のプログラムを組めるエディタと一体となっていて、フルネームは「Z-80エディタアセンブラ・Simple ASM」という。

ディスク版Simple ASMは、マニュアル、サンプルプログラムライブラリー、Z-80ハンドブックの3冊と本体の入ったディスク(エディタアセンブラ以外に開発用ツール付き)が付いて、定価3万9800円だった。

そのSimple ASMが、タケルを通して再販売された。エディタアセンブラ、リロケータブルデバッガ他2本の入ったディスク1枚と、旧販売版なみのマニュアルが付いて5千円だ。

■体験版Simple ASM

今月の付録ディスクでは、コーラルさんのご厚意により、このSimple ASMの体験版を収録している。くわしい使い方は今月号ではできなかったが、とりあえず試してみてほしい。

体験版Simple ASM関連の収録ファイルはぜんぶで5本。

①SIMPLEAS. M
②SIMPLE　. MCH
③GENBIN　. BAS
④GENCOM　. BAS
⑤SIMPLE13. DOC

①②が本体。⑤は体験版に関するかんたんな文書ファイル。③はSimple ASMで作成したHEXファイルをBINファイルに変換するユーティリティ。④はおなじくHEXファイルをCOMファイルに変換するユーティリティだ。

BASICで、付録ディスクを入れ

```
** Simple ASM Ver 1.3 **
  Copyright 1986 by Coral

>1000   LD     A,(HL)
>1010   CP     00H
>1020   JR     Z,EXIT
>1030   CALL   CHPUT
>1040   INC    HL
>1050   JR     LOOP
>■

LOAD    FILES    goto    list    run
```

⇧Simple ASM体験版の画面。BASIC感覚でマシン語プログラミング

RUN"SIMPLEASM

を実行すれば、エディタに入る。

エディタでは、BASICのように、行番号付きでアセンブリ言語のプログラムを入力、編集できる。

また、NEW、AUTO、LIST、LLIST、RENUM、DELETE、SAVE(コマンドだけを実行し、後からファイル名を入力。拡張子SIMが付く。以下のコマンドも同様)、LOAD、MERGEがほぼBASICと同様の使い方で使える。

ほかにも便利なコマンドがあるが、今回は紹介する余裕がない。

また、エディットが終わったら、「AI」と入力してリターンキーを押せば、

```
>LIST
 1000   LD     A,(HL)
 1010   CP     00H
 1020   JR     Z,EXIT
 1030   CALL   CHPUT
 1040   INC    HL
 1050   JR     LOOP
>A
PASS-2
 1000      0000 7E          LD    A,(HL
 1010      0001 FE00        CP    00H
 1020 U    0003 28FB        JR    Z,EXI
 1030 U    0005 CD0000      CALL  CHPUT
 1040      0008 23          INC   HL
 1050 U    0009 18F5        JR    LOOP
PASS-2 END
>■
LOAD    FILES    goto    list    run
```

⇧作成したプログラムをアセンブルしたところ。マシン語コードが表示された。

アセンブルした結果をディスクに書きこんでくれる(拡張子HEX)。Simple ASMから抜けてBASICに戻るには、「BA」だ。

GENBIN. BASを実行すれば、あらかじめ作成しておいたHEXファイルを、BINファイル(通常のマシン語ファイル)に変換してくれる。

■本格的マシン語講座予告

9月号から、このSimple ASMを使ったマシン語講座を開始する。とりあえずは、アセンブラの使い方などからはじめ、徐々にマシン語の奥義に迫りたい。担当は、MSXがなくなったらパソコンをやめると宣言している某大学院生「MSX命」だ。

◆ はじめに　この度はこのソフトをご利用いただき、誠にありがとうございます。この取扱説明書をよく読んで、素晴らしい絵を描いてください。

平成4年1月　作者

◆ 目次

基本操作

◆ 起動の仕方
ディスクドライブにディスクをセットして、CG-7.FD8のプログラムを実行してください。タイトルが表示されて、自動的にリスト2をロードします。入力機器選択画面になりますので、マウス、キーボードをテンキーで選んでください。ターボRの場合は、自動的に高速モードになります。

◆ 終了の仕方
・　電源を切る場合は、ディスケットを抜いてから電源を切ってください。
・　BASICに戻る場合は、必ず「鑑賞」を選んでからESCキーを押して終了してください。

2

◆ カーソルの操作方法
・　キーボード
　カーソルキーで動かします。速く動かしたい場合は、1～0のテンキーを押してください。押した数のドット単位(0の場合は10ドット)にカーソルが動きます。
・　マウス
　マウスを動かすと、その通りにカーソルが移動します。机の端などに行ってしまったときは、一度マウスを持ち上げてください。

◆ 各キーの説明
・　Aボタン(スペースキー)
　各種処理等を、決定・実行するときに押します。
・　Bボタン(GRPキー)
　各種処理等を、取消・中止するときに押します。

◆ メニューの選び方
画面には常にメニューが表示されています。やる処理を選びます。アイコン上にカーソルを移動してAボタンで決定します。枠が移動します。

◆ パレット
メニューの上部にはパレットがあります。左のカラーパレットから描画色を選びます。描画色は真ん中の描画色表示エリアに表示されます。右上のカラーバーは、描画色の色合いを変えます。R、G、Bの値を8段階で変更できます。右は背景色表示エリアです。ここでAボタンを押すと、描画色が背景色になります。

◆ 線の太さ設定
メニューの下の1～6の数字を選ぶと、描画する線の太さを6段階で変えることが出来ます。枠がついているのが選んでいる太さです。

◆ メニューの移動
メニューエリアの中(カーソルが矢印に変化)の何もないところでAボタンを押すと、メニューの表示位置を変えることが出来ます。カーソルを移動させてもう一度Aボタンを押してください。

3

スクロール(画面をスクロールします)
・まず基準となる点を指定します。
・次に基準となる点の移動先を指定します。

グリッド(描画色、太さで指定された間隔で方眼を書きます)

画面クリア(周辺色で画面を消去します)

消しゴム(指定した太さの範囲を1つ前の画面に戻します)
・Aボタンを押しながらカーソルを動かすとその周りを1つ前の画面に戻します。
※アンドゥ参照

アンドゥ(1つ前の画面に戻します)
※1度戻すとそれ以上は戻りません。
※戻るのは、スクロール、グリッド、画面クリアはメニューで選んだ直前、これ以外は第1点を指定した直前です。

6

セーブ(絵をディスクに保存します)

ロード(ディスクから絵を呼び出します)

合成ロード(ディスクから絵の背景色以外を呼び出します)

ファイル名を8文字以内の英数字か片仮名で入力してリターンキーを押してください。
・取り消す場合は何も入力せずにリターンキーを押してください。
※保存、呼び出すディスクは、メニューで選ぶ前にセットしてください。

※エラーが出たら下を見てください。

鑑賞(絵を鑑賞します)
・AボタンかBボタンを押すと戻ります。
・ESCキーを押すとプログラムを終了してBASIC画面になります。

◆ エラーメッセージ
BAD FILE NAME
　ファイル名がおかしいので入力しなおしてください。
DISK FULL
　ディスクがいっぱいです。新しいディスクに変えてください。
DISK OFFLINE
　ディスクがセットされていません。ディスクを確かめてください。
FILE NOT FOUND
　指定されたファイルが見つかりません。ファイル名を入力し直してください。
TOO MANY FILES
　ファイルの数が112を越えました。新しいディスクに変えてください。
DISK WRITE PROTECTED
　プロテクトタブが開いています。書きこみ可にしてください。
DISK ERROR
　上記以外のエラーが発生しました。ディスクを確かめてください。

7

BASICテクニック

DOSに関する13項目

MSX-DOSは「これがDOSの入っているシステムディスクどす」というダジャレのネタ以外に、なんに使えるのか。そもそもいったい何者なのか。そんな根源的な疑問に迫ってみた。

なくてもいいような気もするけれど、なくてはできないこともある

DOSがないと、どういうことで困るのか。

たとえば、現在のMファンの付録ディスクは、いったんMSX-DOS(MSX2/2+の場合はDOS1、ターボRの場合はDOS2)で起動し、それぞれの機種のBASICに移行してから、すべてがはじまる。

このとき、DOSは、「AUTOEXEC.XB」というBASICのプログラムを読みこんで実行する指令を出す役目しかない。たったそれだけのためにDOSを入れているのか。

もちろん、そんなことはない。

では、なんのために?

第1に、圧縮して収録しているファイルを解凍するためのプログラムPMextが、DOSを必要としている。第2に、付録ディスク収録のファイルを別のディスクに手作業でコピーするときはDOSでやったほうがかんたんにできる。第3に、将来、また別の用途が出てくるにちがいない。

DOS1とDOS2の2種類があってちょっと話がややこしい。そこで、今回は、話を比較的単純なDOS1にかぎって、その正体と活用法に迫る。

2 「DOSを起動する」とか「DOSを立ち上げる」とかするには、①DOSシステムの入ったディスクをセットして、②リセットする

DOSだって、プログラムなので、DOSの本体であるプログラムがなければ、そもそもDOSを使うことができない。DOSの本体であるプログラムとは、

MSXDOS. SYS(基本システム)
COMMAND. COM(コマンド実行用)

の2本のファイルのことだ(ターボRは標準モードの場合がこれ。高速モードだと、MSXDOS2. SYSとCOMMAND2. COMの2本になる)。

この2本のプログラムの入ったディスクは、MSX-DOSのシステムが入ったディスクなので、システムディスクという。

システムディスクをセットして、リセットしたり電源を入れたりすると、右ページの項目3のような画面になり、DOSが起動する。もしディスクのなかに「AUTOEXEC. BAT」という自動実行バッチファイル(33ページ項目11参照)があれば、続いてそのファイルも自動的に実行するという楽しい機能もある。付録ディスクもその機能を使っているのだ。

1 DOSはDisk Operation Systemの略っていうくらいだから、ディスクを使うためのものであることくらいはだれでもわかる

DOSのフルネームはDisk Operation System(ディスク・オペレーション・システム)。DOSはディスクを操作するためのシステムらしいということが、まず最初にわかる。

ディスクを操作するために、DOSは、まず、ディスクを読み書きできる状態にする機能がある。これを、フォーマットという。

フォーマットまえのディスクはただの磁性体の円盤にすぎない。DOSは、原野のようなディスクを切り開き、きれいに区画整理し、ディスク全体を機能的な記憶媒体に仕立てあげる。BASICのフォーマットも、じつはMSX-DOSのフォーマットなのだ。

このフォーマットによって、DOSを通して、ディスクのなかにいろいろなファイルを作ったり、消したり、複写したりすることができるようになる。

しかも、このフォーマットは、PC98などの16ビット、32ビット機で広く使われているMS-DOSとも共通なので、MSX用のディスクのファイル管理をPC98でおこなったりもできる。

じっさい、Mファンでも付録ディスクの制作中はPC98でファイルを管理しているのだ。

3 BASICの「Ok」に相当するDOSの「A>」は「Aプロンプト」と読む。「A」はカレントドライブ名、「>」がプロンプトだ

BASICの画面では、入力する行の先頭にはカーソルしかない。そのかわりといってはなんだが、その直前の行に「Ok」という文字が表示されている。この「Ok」を「ぼよよん」とかに設定している人もいるかもしれないが、ともかくこれをプロンプトという。

DOSにも形はちがうけれど、おなじようなものがある。それが「A>」。

厳密にいうと、「>」だけがプロンプトで、Aはカレントドライブを表示しているだけ。カレントドライブとは、拒食症で死んだカーペンターズのボーカルとドライブすることではないし、可憐なトドが生きているわけでもない。ドライブ名を省略したときに設定されるドライブのことをカレントドライブというのだ。試しに「B：」と入力してリターンキーを押すと、以後、「B>」というプロンプトになり、ドライブ名を省略してさまざまなことをすると、ドライブBに入ったディスクに対して動作するようになる(ただし、1ドライブの場合、ドライブBはおなじドライブ)。

```
MSX-DOS version 1.03
Copyright 1984 by Microsoft

COMMAND version 1.11

A>■
```

DOSの初期画面。「A>」で始まる行がコマンドライン

4 DOSコマンド「BASIC」でDOSからBASICへ行けば、BASICの「CALL SYSTEM」でDOSにもどれる

DOSは、BASICの本体が入っているROM(32Kバイト)を切り離し、そのぶんRAMを64Kバイトにしている。DOSのシステムや、COMMAND.COMなどもRAMに入っているのだ。このメモリ構成では、とうぜんBASICのプログラムは動かない。

しかし、DOSの基本コマンドのなかに「BASIC」という、そのままの名前の命令がある。このコマンドを実行すると、BASICの入っているROMが使用可能になり、BASICの初期画面に移行し、BASICのバージョンやコピーライト表示など、どう見てもいつものBASICとおなじものが現れる。しかし、いつものBASICにない特徴が1つだけある。

それは、システムディスクをセットしたまま、「CALL SYSTEM」というBASICの命令を実行すれば、ふたたびDOSにもどるということだ。そうでないときにこの命令を実行すると、エラーになってしまう(ターボRの高速モードではつねに有効)。

```
MSX BASIC version 2.0
Copyright 1985 by Microsoft
23430 Bytes free
Disk BASIC version 1.0
Ok
call system

A>■
```

BASICからCALL SYSTEMでDOSにもどった

5 DOSの基本命令はぜんぶでたったの13種類しかないが、COMファイルを増やしていけば使用可能コマンドは増殖していく

DOSの基本コマンドは、ほんとうに基本的なものばかりで、フォーマット、日付・時間の設定、ファイル一覧の表示、テキストファイルの表示、ファイルの複写・削除など表にまとめてみると意外に少なく、13種類しかない(ただしこれはDOS 1の場合)。

しかし、拡張子が「COM」になっている実行可能なプログラム(COMファイル)を増やせば、そのファイル名の入力だけでコマンドを実行できる。こうした拡張コマンドを集めたものが、『MSX-DOS TOOLS』(アスキー/1万4800円)というユーティリティ集だ。アセンブラなどでこのCOMファイルを自作することも可能なのだ。

◉MSX-DOSコマンド簡易一覧表　※f〜はドライブ名を含むファイルスペック

コマンド	書式　※［　］は省略可	機能［書式の省略項目の機能］
BASIC	BASIC　[f1]	BASICに行く[その後、f1を読みこみ、実行する]
COPY	COPY　f1　f2 COPY　f1+f2……　f0	f1をf2へ複写 f1、f2……を結合してf0へ複写
DATE	DATE　[yy/mm/dd] ※「／」は「ー」でも可。yyは年(4桁でも可。ただし1980〜2079年に限る)、mmは月、ddは日	内蔵カレンダの日付を表示し、現在の日付を入力する［直接、日付を設定する］
DEL (=ERASE)	DEL f1 (=ERASE f1)	f1を削除(「DEL」のかわりに「ERASE」を使ってもおなじ)
DIR	DIR　[/P]　[/W]	ファイル一覧を表示［画面いっぱいに表示したらキー入力待ち］［ファイル名のみを表示］
FORMAT	FORMAT	ディスクのフォーマット
MODE	MODE　n　※n：1〜80	表示文字幅の設定
PAUSE	PAUSE　[〈メッセージ〉]	バッチ・ファイルの一時停止とキー入力待ち［メッセージの表示］
REM	REM　[〈コメント〉]	バッチ・ファイル中に注釈を書く
REN (RENAME)	REN　f1　〈新ファイル名〉 (RENAMEも同様)	f1のファイル名を新ファイル名に変更
TIME	TIME　[hh：mm：ss] ※hhは時、mmは分、ssは秒。　入力の最後にa(午前)かp(午後)を付ければ12時間制になる	内蔵時計の時刻を表示し、現在の時刻を入力する［時刻を設定する］
TYPE	TYPE　f1	f1の中身を画面に表示する
VERIFY	VERIFY ON VERIFY OFF	ディスク書きこみ時に書きこんだ内容をチェックする機能のON、OFF切り換え。最初はOFF

※この表は、MSX-DOS 1のコマンドを一覧するためのものなので、実行するときの条件などは省略しています。よりくわしい解説は、各機種に付属しているマニュアルを参照してください。また、VERIFYコマンドは、ディスクドライブの機種によっては使用できないものがあります。なお、おもにターボRの高速モードで使用されるMSX-DOS2はこの表のコマンドも含みますが、表示などが多少ちがいます。

6 コマンドを入力するコマンドラインの裏に、前回入力したコマンドを記憶している場所がある。これをテンプレートという

DOSは、BASICのときとちがって、1行単位で独立しているので最初はちょっと使いづらい。しかし、前回入力した文字列はテンプレートと呼ばれるところに記憶されていて利用できるようになっている。

たとえば、

A>TYPE　AUTOEXEZ.BAT

と入力して、ファイルがないというエラーが出たとする(CをZと打ち間違えた)。すると次は、SELECTキーを押したあとにZキーを押すことで、下図のように文字Zの直前までの部分を一気にテンプレートから呼びもどすことができる。そこで前回入力ミスしたCだけを打ちこむ。そして、カーソルキーの下を押すと、残りの「.BAT」がテンプレートからやってきて完成。

ほかにもDELキーで1字スキップ、カーソルキーの右で1字ずつコピー、上でコマンド取消……など便利な機能がある。

◉テンプレートとコマンドライン

テンプレート(見えない)	TYPE AUTOEXEZ.BAT
	↓ SELECT+ZでZの手前まで一気にコピー ↓
コマンドライン	TYPE AUTOEXE

7 たいていのファイル操作は、BASICでもできるけど、DOSでやったほうがかんたんで速い。その代表が、COPYコマンドだ

Disk BASICは、基本的にDOSのBASIC出張所みたいなものなので、DOSのファイル操作はほとんどBASICでもできる。

しかし、BASICではファイル名をダブルクォート(「"」)で囲わなければいけないなど、操作感覚的にはDOSのほうがやりやすい。

それになんといっても、1ドライブでファイルを複写するときはメモリを広く使っているDOSに限る。DOSでは短いプログラムなら数本まとめて1回でコピーしてくれるのだ。

```
A>copy *.fmx b:
DYNAMIC  FMX
ZANZIBER FMX
OUTLAW   FMX
DOOM     FMX
RIKUGUN  FMX

Insert diskette for drive B:
and strike a key when ready■
```

⊕数ファイルぶんをいっきにコピーする

8 DIRコマンドでファイル一覧を見れば、いつ作ったファイルか、どのくらいの大きさのファイルなのか、ひと目でわかる仕組み

BASICではまったく利用できないのがDOSのDIRコマンドで見るファイル情報だ。ファイル名のあとにそのファイルの大きさ(バイト数)、ファイルを作成・更新した年月日時間(タイムスタンプ。時間は末尾のa、pで午前、午後を表す)まで知ることができる。ファイルが多くて1画面に表示しきれないときは、「DIR /P」と入力すれば下の画面のようにとちゅうでキー入力待ちになるし、ファイル名だけでいいときは「DIR　/W」と入力すればいい。

```
MSXDOS    SYS     2432 85-08-23  9:29p
COMMAND   COM     6656 85-09-02 10:10p
MSXDOS2   SYS     4870 90-09-03  4:58p
COMMAND2  COM    15708 91-06-19  8:11p
AUTOEXEC  BAT       18 92-03-31  3:08a
AUTOEXEC  XB       298 92-06-06  4:30p
LOADER    BIN      391 92-07-29  5:46p
MSXFAN    BIN    11146 92-07-29  5:46p
FONT12    XBN    58884 92-06-05 11:56p
PALETTE   BIN     1024 92-07-29  8:27p
TITLE     XB      1165 92-07-29 11:03p
TITLE'10  XS7    25754 92-07-28  6:29p
TITLE'10  BGM     1984 92-04-22  3:30p
MENU      XB      1850 92-08-03  3:10p
MENU'10   XP5    14625 92-08-01 12:26a
MENU'10   XTB     8183 92-08-04  6:50p
TYPE-A    XB      2867 92-07-07 12:22p
TYPE-C    XB      2023 92-07-06 12:24a
TYPE-FM   XB      2991 92-07-31  9:23a
RUN-XX    XTX     2281 92-07-31  5:25a
GAME      XB       236 92-08-01
DEMODATA  BIN      519 74-01-02 12:56a
PREPROG   BIN      824 26-04-29  8:15p
Strike a key when ready ■
```

◐「DIR /P」を実行したときの画面。なにかキーを押すと次を表示する

```
KANJI     BAS     KANA      BAS
EX        BAT     RUN-CG    XB
CG10      LZH     GAMEJ'10  C
GAMEJ'10  XS7     FM-MS'10  A
DYNAMIC   FMX     ZANZIBER  FMX
OUTLAW    FMX     DOOM      FMX
RIKUGUN   FMX     AV-OCT    AVX
AV-MAC    AV      AV-LINE   AV
GRP-1     AV      GRP-2     AV
RUN-PSTN  XB      PMEXT2    DOC
PMEXT     COM     TIMERBAL  PMA
ALLZERO   PMA     KAIJU2    LZH
HUSTLE    LZH     OTHELLO   LZH
BW100     PMA     ATE-Q'10  XB
ATEQ1'10  XS7     ATEQ2'10  XS7
RUN-MIDI  XB      MIDISFX   COM
TYPE-M    XB      OMAKE     A
UEDIT     COM     FONT1     CHR
DEMO      BAS     FORELLE   BIN
LOGO      BIN     BGM       OBJ
B-CLN'10  C       C-CLN'10  C
D-CLN'10  C
       93 files      3072 bytes free
A>■
```

◐「DIR /W」を実行したときの画面。BASICのFILESと似ている

9 文字列を知的に省略できる「*」や「?」。ちっとも野蛮じゃないし、カードでもないのに、ワイルドカードとはこれいかに

ファイル操作のときに知っておかないとものすごく損するものがこのワイルドカードだ。もともとトランプ用語で、どんなカードとしても使えるカードのことだが、ファイル関係では1文字ぶんと置き換える「?」とそれ以降の文字列ぜんぶと置き換える「*」の2種類がある。

ただし、ファイル名部分と拡張子部分はピリオドで厳密に区別され、このピリオドはワイルドカードでは置換できない点に注意。

下にいくつかの実例を示した。いろいろ試して覚えよう。

```
A>DIR D*.FM?
DYNAMIC   FMX     2537 92-07-24 11:10p
DOOM      FMX     3059 92-07-25  1:45a
        2 files      3072 bytes free
A>DIR ?.*
File not found
A>DIR ??.*
EX        BAT       15 92-07-31  8:36p
        1 file       3072 bytes free
A>DIR .BAT
AUTOEXEC  BAT       18 92-03-31  3:08a
DEMO      BAT      384 92-08-02 12:18a
EX        BAT       15 92-07-31  8:36p
        3 files      3072 bytes free
A>DIR MSX*.*
MSXDOS    SYS     2432 85-08-23  9:29p
MSXDOS2   SYS     4870 90-09-03  4:58p
MSXFAN    BIN    11146 92-07-29  5:46p
        3 files      3072 bytes free
A>■
```

⊕ワイルドカードを使うとファイル探しも楽

10 周辺装置(デバイス)をファイルのように扱うDOSマジック。デバイス・ファイルは、「CON」と「PRN」を知っていればOK

ふつうのファイル名には使えない名前が、「CON」、「PRN」、「LST」、「AUX」、「NUL」だ。いずれも、周辺装置をファイルとして扱うときの名前で、順にコンソール(画面とキーボード)、プリンタ、プリンタ、補助入出力装置(モデムなど。ただし、起動時はNULとおなじ)、ダミーを意味する。このうち、CONとPRNが最重要だ。たとえば、あるファイル(ABC.TXTとする)をプリンタに打ち出すときは、「COPY　ABC.TXT　PRN」を入力すればいいのだ。CONについては次項。

11 DOSでもBASICみたいにかんたんにプログラムが組める。DOSの機能を存分に活用する、やさしいバッチ・ファイルの作り方

DOSでいちばんおもしろいのが、バッチ・ファイル。バッチというのは、「ひとかたまり」といったていどの意味で、バッチ・ファイルは、いくつかのコマンドをひとかたまりにしてファイルにしたもの。拡張子を「BAT」にしておくと、バッチ・ファイルのファイル名を入力するだけで、そのファイルのなかに入っているコマンドを順に実行してくれる仕組みなのだ。

ちなみに付録ディスクなどに入っている「AUTOEXEC. BAT」は、自動実行バッチ・ファイルという。この名前のバッチ・ファイルは、DOSが起動した直後、自動的に実行されるのだ。BASICの「AUTOEXEC. BAS」と似ている。

■DEMO .BAT

```
A>copy con demo.bat
█
```

⬆バッチ・ファイル作成開始

```
A>copy con demo.bat
mode 40
pause 40ジ゛モ－ド゛。 screen0 ト オナジ゛
mode 32
pause 32ジ゛ モ－ド゛。 SCREEN 1 ト オナジ゛
mode 80
█
```

⬆コマンドごとにリターンキーを押して入力していく

```
A>copy con demo.bat
mode 40
pause 40ジ゛モ－ド゛。 screen0 ト オナジ゛
mode 32
pause 32ジ゛ モ－ド゛。 SCREEN 1 ト オナジ゛
mode 80
pause 80ジ゛ モ－ド゛。 screen0 / width80 ト オナジ

mode 40
pause DOS ヲ ツカウトキハ ヒヅ゛ケ ト ジ゛カン ヲ セッテイ シマ
ショウ
pause マズ゛ ヒヅ゛ケ
date
pause ツギ゛ニ ジ゛カン
time
pause ファイルイチラン (デ゛ィレクトリ) ヲ ヒョウジ゛ シマス。 1
ガ゛メン ヲ コエルトキハ キ－ニュウリョク マチニ ナリマス
dir /p
pause コンド゛ハ ファイルスペ゜ック ダ゛ケ ノ ヒョウジ゛ デ゛ス
dir /w
pause コレデ゛ オワリデ゛ス
^Z
        1 file copied
A>█
```

⬆最後にCTRL+Zで「^ Z」を入力して作成終了

■EX .BAT

```
basic %1.bas
```

⬆「EX. BAT」の中身はたったこれだけ。%1が仮引数

```
10 IF PEEK(&H2D)<2 THEN CLS:CALL SYSTEM
20 CALL KANJI1
30 CALL SYSTEM
```

⬆%1が「KANJI」のときに実行される「KANJI. BAS」

```
10 IF PEEK(&H2D)<2 THEN CLS:CALL SYSTEM
20 CALL ANK
25 CLS
30 CALL SYSTEM
```

⬆%1が「KANA」のときに実行される「KANA. BAS」

■バッチファイルの作り方

基本的には、①拡張子をかならず「BAT」にする、②実行したい順にコマンドをならべる、の2点を守って、テキストファイルを作ればいい。テキストファイルの作り方もいろいろで、エディタを使う、プログラムでファイルを作る、などがあるが、デバイス・ファイルの「CON」を利用して直接バッチ・ファイルを作る方法がある。

COPY CON 〈ファイル名〉

を実行すると、それ以降、CTRL＋Zを入力するまでに入力した文字を指定した名前でファイルにすることができるのだ(上の写真)。

付録ディスクのDOSコーナーに行き、コマンドラインから、

DEMO

と入力してリターンキーを押しバッチ・ファイルを体験してみよう。

さらに仮引数というのは、このバッチ・ファイルを実行するときに打ちこむファイル名の横に9個まで引数をオプションで付けられる機能だ。この引数は、バッチ・ファイルのなかでは、順に％1～％9という名前で(ちょうど文字変数のように)使われる。この仮引数を使ったサンプルが上記のEX. BAT(漢字モード、かなモードを切り換える)だ。付録ディスクに入っているので、MSX2＋以降のユーザー(日本語カートリッジでも可)は、ぜひ「EX KANJI」と「EX KANA」も試してほしい。

12 DOSを使っているときになにか悪いことがあると出るメッセージ。メッセージたちが伝えようとしていることはなにか

エラーメッセージを大きく3つにわけると①コマンド名そのものの間違い、②コマンドを実行しようとして問題にぶつかった、③コマンドの書式の間違い、の3つになる。

まず、①の場合には、次のメッセージが出る。

Bad command or file name

これは、コマンド名かファイル名がまちがっている(指定されたコマンド用のファイルがない)と訴えていて、入力ミスであることが多い。

②でよくあるのは(DIRコマンドのときなど)、

File not found

これは、いまセットされているディスクには指定されたファイルがない、ということを意味している。ディスクの入れ間違いもけっこう多い。

Insufficient disk space

Not enough memory

この2つは、上がディスクの残り容量、下がメモリの容量がたりないといっている。上はディスクを取り替えればいいけれど、下はプログラムの問題なので難しい。

ほかにも、Write error(書きこみ失敗)やRename error(ファイル名変更に失敗)などがあるが、どれも原因は意外と単純なはずだ。

③のタイプでは、Invalid～ではじまるメッセージが多い。これはようするに、コマンドに付けた引数に間違いがある場合で、数値が範囲を超えていたり、指定したドライブ名が間違っていたりしたときの注意なので、いま入力したものの引数を注意深く見直そう。

```
A>BASOC
Bad command or file name
A>█
```

⬅コマンドの入力ミスでよく出るエラー。BASICのSyntax errorのようなもの

13 節目、折り目で、DOSがユーザーの反応を待っている7つのメッセージ。準備が整いしだい、このメッセージにはこのキーを押す

```
Write protect error writing drive A
Abort, Retry, Ignore? █
```

⬆3つの選択を迫るメッセージ。頭文字で答える

入力待ちメッセージの代表は、「Strike any key when ready」(準備ができたらなにかキーを押してください)。これはそのとおりに、なにかキーを押せばいい。同様のものに「Press any key for retry」がある。

「Enter～」で始まるものは、日付または時刻の入力待ちだ。

ちょっと難しいのは、「Abort, Retry,Ignore?」(中止する、もう一度おなじことをやる、エラーを無視する？)。3つの選択肢のうち1つを大文字になっている文字キーを押して選ぶ。ディスクのセットミスなどに多いので、確認して、もう一度やり直すことが多い。

また、たとえばファイルを全部削除しようとしたときには、「Are you sure(Y／N)？」(ほんとうにいいんですね？)と念を押してくる。いいんだよ、ならY、あっやっぱりやめ、ならN。同様のものに、「Terminate batch file(Y／N)？」(バッチファイルを終了させますか？)がある。

BASICテクニック

キーボードの達人になれる7つの秘術

地、水、火、風が万物の根源なら、ソフトの根源はキーボードだ。昔、わしらがまだ若いころ、ファンダムの作品もAVフォーラムの作品もキーボードから1字1字入力されたものじゃった。

パソコンはどんどんアナログでファジーでビジュアルでダイレクトになってきた。

入力装置といえば、かつてはデジタルばかりだったが、現在は、勢力を拡大しつづけるマウスがアナログ的な入力を普及させてしまった。そして、ファイルの存在や性質が絵(アイコン)で表され、その絵に対してボタンを押したりカードをめくったりするような感覚で操作するものが主流になりつつある。

ルイス・フロイスではないが、きくところによれば、最近では画面に対して直接働きかけるパソコン用ペン(ライトペンではない)が脚光を浴びているとか。赤ちゃんが生まれて2本足で立ち上がるまでくらいの近い将来、そのペンだけを備えたポケットコンピュータが本気で流行しているかもしれない。

すると、キーボードも、いずれは蒸気機関車やハエ取り紙やロバのパン屋や蚊帳(かや)やメンコのように、ノスタルジーの対象になっていくのだろうか。

いや、まさかペンやマウスだけでプログラムを組んだり、文書入力したりするやつは10年たっても出てこないだろう。キーボードは、まだとうぶん、基本的な入力装置でありつづける。

根源のキーボードにまつわる7つの秘術をまとめてみた。

1 どんな文字でも飲みこむ LINEINPUT文の使い道

プログラムをはじめて組もうとしたときにまず使うキー入力関係の命令といえば、「INPUT」だろう。しかし、いろいろなプロンプト文をはさんで使っていると、ちょっと不都合なことに出会う。

たとえば、

INPUT ″ニュウリョクシテクダサイ″；A$

のような命令文を実行すると、

ニュウリョクシテクダサイ？ ■

としかならなくて、ガクッとくる。この「？」は取るに取れない。

もう1つ、より本質的に困るのが、「,」と「″」の入力だ。どちらの記号も特殊な機能を持っているために入力文字列のなかで共存できないのだ。

この2つの問題を同時に解決するのが「LINEINPUT」という命令だ。変数は文字変数のみ、カンマで複数の変数を区切ることができないという2点を除けば、INPUT文と使い方はおなじ。

プロンプト文には「？」や空白が付かないし、「,」や「″」を含めてリターンキーを押すまでに入力したすべての文字(254字を超えたぶんは無視する。また、文字列領域を確保しておかないとエラーになる)を受け入れてくれる。

リスト1は、そのかんたんな使用例。ここでは、画面の中央に表示しなおしているだけだが、そのかわりにディスクに書きこんだりすれば日記帳ができるわけだ。

◉リスト1　LINEINPUTのサンプル

```
10 CLEAR 300
20 COLOR 15,4,7:SCREEN 1:WIDTH 29
30 LINEINPUT "MESSAGE?(MAX254)>>> ";A$
40 LOCATE 0,10:PRINT "MESSAGE= ";A$
```

2 関数INKEY$とBIOSの&H156

INKEY$を使っていると、無意識に触れたキーが入力としてキーバッファにたまり、いざというときにINKEY$でひろわれて誤動作を起こすことがある。

キーバッファとは、キー入力の情報を連続して記録しているところで、ここにたまったものが1字ずつ順にINKEY$に渡されていくのだ。

そこで、不用なキー入力をてきとうな時期に掃除する必要がある。これがキーバッファクリアだ。

これには2とおり方法があって、1つはリスト2－Bの行40のように毎回よぶんなキー入力を使いはたしてから次に進むBASICによる方法。INKEY$を1回使うたびにキーバッファにたまった入力は1字ぶん減るのだ。

もう1つは、リスト2－Aの行10と行40のように、USR関数を＆H156（または10進数で342）に定義しておいて、それを呼び出すという方法。このアドレスは「キーバッファクリア」という、まさにそのまんまのBIOSのアドレスなのだ。

ただし、INKEY$の直前や直後に置くと入力がきかなくなったり、反応が悪くなったりすることがあるので注意しよう。

●リスト2-A BIOSを使ったキーバッファクリア

```
10 DEFUSR=&H156 'or 342
20 A$=INKEY$:IF A$="" THEN 20
30 BEEP
40 A=USR(0) 'ｷｰﾊﾞｯﾌｧｸﾘｱ
50 GOTO 20
```

●リスト2-B BASICだけのキーバッファクリア

```
10 '
20 A$=INKEY$:IF A$="" THEN 20
30 BEEP
40 IF INKEY$<>"" THEN 40 'ｷｰﾊﾞｯﾌｧｸﾘｱ
50 GOTO 20
```

3 ファンクションキーで和音を演奏する方法

ファンクションキーでは次ページの5で扱う登録機能のほかに、割りこみという技が使える。

ON KEY GOSUB a, b, c, d, e, f, g, h, i, j（a～jはそれぞれF1～F10が押されたときの飛び先行番号。使わないのはカンマだけか省略）という書式で定義して、

KEY(n)ON （nは割りこみ許可するキー番号）

で割りこみ許可をすれば、プログラム実行中ならいつでも、ファンクションキーの入力に応じて指示されたサブルーチンに飛ばせる。

この機能は付録ディスクのAVフォーラムで多用されているが、ほかにもリスト3のようにして和音演奏機を作ったりなどけっこう使い道がある。

```
[F 1] [F 2] [F 3] [F 4] [F 5]
  C     F     G    Am    Em
```

↑リスト3実行画面。F1～F5にC～Emが対応。1回押すと4分音符ぶん和音が鳴る

●リスト3 簡易和音演奏機

```
10 COLOR 15,4:SCREEN 0:KEYOFF:WIDTH 40
20 PRINT"[F 1] [F 2] [F 3] [F 4] [F 5]"
30 PRINT"  C     F     G    Am    Em "
40 M$="V15T LOS11M10000":PLAY M$,M$,M$
50 ON KEY GOSUB 100,110,120,130,140
60 FOR I=1 TO 5:KEY(I) ON:NEXT
70 '
80 GOTO 80 '<<<<< MAIN!
90 '
100 PLAY "C","E","G":RETURN
110 PLAY "C","F","A":RETURN
120 PLAY "D","G","B":RETURN
130 PLAY "C","E","A":RETURN
140 PLAY "E","G","B":RETURN
```

4 CTRL+STOPより強い命令

マシン語では、よくリセットするしかないバグというのがあるが、BASICでは、バグがあるとかってに実行をやめてエラーメッセージを出す。もし、無限ループに入ってしまっても、だれもが知っている強制終了の方法がある。それが、CTRL+STOPだ。

しかし、このCTRL+STOPという キー入力を感知して働く割りこみ命令がある。それが、

ON STOP GOSUB～

だ。これも、上の書式で飛び先行番号を定義したうえで、

STOP ON

で割りこみ許可をする。ちなみに、BASICの割りこみ命令では一般に、～ONが割りこみ許可、～OFFが割りこみ禁止、～STOPが割りこみ保留（キー入力は記憶しているが割りこみ先へ飛ぶのを保留する）となっている。

これを使うと、リスト4のように、CTRL+STOPでプログラムの実行を止めようとするユーザーに対して、なんらかのメッセージを伝えることができる。しかし、これはCTRL+STOPの強制終了の機能をなくしているので、へたをするとリセットする以外に終われないBASICプログラムができてしまうことがある。

```
ﾔﾒﾁｬｳﾉｵ(Y/N)?
```

↑CTRL+STOPするとメッセージを表示。YかYを入力すると終了する

●リスト4 終わりたがらないプログラム

```
10 COLOR 15,4,7:SCREEN 0:WIDTH40:KEYOFF
20 ON STOP GOSUB 40:STOP ON
30 CLS:PRINT "NOW WORKING...":GOTO 30
40 DEFUSR=342:A=USR(0)
50 LOCATE 10,10:PRINT "ﾔﾒﾁｬｳﾉｵ(Y/N)?"
60 A$=INKEY$:IF A$="" THEN 60
70 IF INSTR(" Yy",A$)>1 THEN END
80 PLAY"V15TS1M10000O6E4C2":RUN
```

5 ファンクションキーを手順登録装置にする関数INPUT$

ファンクションキーには、15バイトの文字を登録できる。その文字のなかには、キャラクタコード31以下のいわゆるコントロールコードも含まれる。これを利用して、DELキー13回ぶんとか、自動的に挿入モードになって「LOAD"」を挿入したうえで実行するなど便利な機能を設定できるのだ。

しかし、その場かぎりで使いたい機能の場合、いちいちコントロールコードを調べるのはめんどうだし、設定もたいへんだ。

そういうときには、

KEYn, INPUT$(15)
(nは設定したいキー番号)

を実行して、登録したいキー操作をじっさいにやってみよう(この関数は受け付けた文字を表示しないので頭のなかで想像しながら)。あまったら、ESCキーをずっと押しておく。Okが出たら終了。

関数INPUT$はCTRL+STOP以外の、すべての入力を受けつけ、指定された数ぶんの文字列を作るのだ。

たとえば、CTRL+F、CRTL+E、RETURN(残りはESC)という操作をF1に登録すれば、行番号にあわせてF1キーを押しただけでその行を抹消できる。ほかに、CRTL+F、INS、「'」(文字)、スペース、RETURN(残りはESC)をF1に行番号にあわせてF1キーを押すと、その行はREM文になってしまう……、などなど、いろいろ楽しめる。

6 GRAPHキーをBボタンにする方法

ごくたまにだが、ファンダムでジョイスティック専用という投稿作品を見かけることがある。遊んでみると、カーソルキーとスペースキーでも問題なく遊べそうだ、なぜジョイスティック専用なのだろうと思っていると、リプレイがBボタンだった……というようなことがある。GRAPHキーかなにかをBボタンがわりに使えば、キーボードだけで遊べたのに。

右の表が、そのために必要な資料だ。メモリのアドレス&HFBE5以降はキーマトリクスというワークエリアで、押されているのがどのキー(物理的なキー)かをリアルタイムで示している。

1バイトにつき8つのキーがビットごとに対応し、あるキーが押されていればそのビットは0、押されていなければ1になっている。

あるキーに対応するビットが0になっているかどうかを調べる方法はいくつもあるだろうが、ここでは、論理演算と関係演算を使った方法をGRAPHキーを例にとって、記しておこう。ちょうどSTRIG関数とまったくおなじようにしてGRAPHキーの入力を調べられるのだ。

GRAPHキーに対応するアドレスは&HFBEBのビット2。そこで、手順としては、まずそのアドレスの内容を取り出し(Aと置く)、ビット2の情報だけを抜き出す(Bと置く)。

```
A=PEEK(&HFBEB)
B=A AND 2^2
```

こうすると、GRAPHキーが押されているときは、Bは0になり、押されていないときは2^2つまり4になる。BはANDでマスキングされているので、この2つの値以外にはならない。

そこで、じっさいにキー入力判定に使うときは、STRIG関数の特性にあわせて、次のような関係演算式を作る(TBと置く)

```
TB=(B=0)
```

こうしておくと、GRAPHキーを押しているときは、Bは0になるので、TBは-1になり、押していないときはBは0でないのでTBは0になる。これは、STRIG(3)でBボタンが押されたか押されてないかを調べるときとおなじだ。

じっさいには、いちいち3つに分けずにいっぺんに式を作ることが多い。つまり、

```
TB=(PEEK(&HFBEB)AND4)=0
```

このTBによってGRAPHキーをBボタンふうに使えるのだ。

●図1 キーマトリクス表

ワークエリアのアドレス / ビット

	b7	b6	b5	b4	b3	b2	b1	b0
FBE5	7	6	5	4	3	2	1	0
FBE6	;	[	@	¥	^	-	9	8
FBE7	B	A	_ ※1	/	.	,	]	:
FBE8	J	I	H	G	F	E	D	C
FBE9	R	Q	P	O	N	M	L	K
FBEA	Z	Y	X	W	V	U	T	S
FBEB	F3	F2	F1	かな	CAPS	GRAPH	CTRL	SHIFT
FBEC	RETURN	SELECT	BS	STOP	TAB	ESC	F5	F4
FBED	→	↓	↑	←	DEL	INS	CLS	SPACE

※1 同時にSHIFTキーを押したときの文字(アンダーバー)。このキー単独ではかな以外の文字を表示しない。

【テンキー】

	b7	b6	b5	b4	b3	b2	b1	b0
FBEE	4	3	2	1	0	※2	※2	※2
FBEF	.	,	-	9	8	7	6	5

※2 メーカーによって対応する文字が異なる。

7 究極の打法・ブラインドタッチ

原稿を見ながら演説する政治家はいても、台本を見ながらしゃべる漫才師はいないし、鍵盤を見ながら演奏するピアニストもいない。

見なければならないものがほかにあるからだ。漫才師は聴衆を、ピアニストは楽譜や指揮者を見るのにいそがしい。

ゲームをしながらジョイスティックを見ている人はいないし、CGをかきながらマウスを見ている人もいない。どちらも画面を見るのにいそがしい。

しかし、プログラムなどを入力するときに、入力されたものが次々に表示されている画面を見ないでキーボードを見ていたり、せいぜい、キーボードと画面を交互に見比べていたりする人は多い。

キーボードを見ずにキーを打つ「ブラインドタッチ」という打ち方を知らないからだ。

思い切って、キーボードをまったく見ずに画面やリストだけを見ながらプログラムを打ちこんでみよう。それだけで、思ったよりはうまくいく。自分がキーの配置をけっこう覚えていることに気づくだろう。もっとも特殊な記号やグラフィックキャラクタは別だが。

↑リスト5の実行画面。打ったキーが光る

図2のような指の守備範囲を守れば、指をほとんど動かすことなく楽にキー入力していける。右手小指の守備範囲がいやに広いが、そのかわりこのあたりは使用頻度の低いキーばかりで、おそらく、もっともよく使うのはBSキーだ。

SHIFTキーは、左手の範囲のキーを打つときは右手で、右手の範囲のキーを打つときは左手で押さえるようにする。SHIFTキーが2つあるのはそのためだ。

キーを打ったら、つねにホームポジションと呼ばれる位置に指を持ってきておくのが原則だ。

右手の小指で0を打ったり、左手の薬指でXを打ったりすることに抵抗のある人もいるかもしれないが、ほんのちょっとその違和感をがまんしていると、数日で英数字だけならキーボードをまったく見ないで打てるようになる。それがブラインドタッチというやつだ。

ブラインドタッチに慣れると、頭のなかで描いた文字列がすぐに画面に現れるようになる。

キーボードは他機種でもほとんどおなじなので、MSXでブラインドタッチができると、ほかの機械でもできるようになる。キーボードという文化がしばらく続くかぎり、ここでブラインドタッチの練習をしておいて損はない。

ちなみにリスト5はそのための英数字限定の練習用だが、まず手始めにこのプログラムをキーボードを見ずに打ちこんでみよう。

●図2　ブラインドタッチのための指のポジション

左手　右手

小指　薬指　中指　人差指　人差指　中指　薬指　小指

ESC 1 2 3 4 5 6 7 8 9 0 - ^ ¥ BS
TAB Q W E R T Y U I O P @ [RETURN
CTRL A S D F G H J K L ; :]
SHIFT Z X C V B N M , . / _ SHIFT
CAPS GRAPH SPACE かな

※赤い字のキーがそれぞれの指のホームポジションとなるキー

●リスト5　簡易ブラインドタッチ練習装置

```
100 '*** ｶﾞﾒﾝ ｼｮｷ ｾｯﾃｲ
110 COLOR 15,0,0:SCREEN 5,0
120 PSET (0,0),0:DRAW "S4A0C15"
130 OPEN "GRP:" AS #1
140 '*** ﾍﾝｽｳ ｶﾝｹｲ ｼｮｷ ｾｯﾃｲ
150 DEFINT A-Z
160 DEFFNP(N)=PEEK(&HFCB7+N)+PEEK(&HFCB8
+N)*256   'LP ﾉ Xｻﾞﾋｮｳ ﾄ Yｻﾞﾋｮｳ
170 DIM X(5,7),Y(5,7)
180 K$="R8F2D10G2L8H2U10E2"
190 '*** ｷｰﾎﾞｰﾄﾞ ｻｸｶﾞ
200 FOR I=0 TO 3
210 READ X,Y,L
220 DRAW "BM=X;,=Y;"
230 FOR J=0 TO L:DRAW K$:READ A$
240 AD=VAL(MID$(A$,2,1))
250 B=VAL(MID$(A$,3,1))
260 X(AD,B)=FNP(0)-1:Y(AD,B)=FNP(2)+1
270 DRAW "BF2BD2":PRINT #1,MID$(A$,1,1);
:DRAW "BE4BR2"
280 NEXT:NEXT
290 '*** ﾌﾞﾗｲﾝﾄﾞ ﾀｯﾁ ｱｼｽﾄ
300 FOR AD=0 TO 5:B=7
310 C$=BIN$(256+PEEK(&HFBE5+AD))
320 C=INSTR(9-B,C$,"0"):IF C=0 THEN 350
330 B=9-C:LINE (X(AD,B),Y(AD,B))-STEP(10
,12),15,BF,XOR:BEEP:LINE (X(AD,B),Y(AD,B
))-STEP(10,12),15,BF,XOR
340 B=B-1:GOTO 320
350 NEXT
360 GOTO 290
370 '*** ｻｸｶﾞﾃﾞｰﾀ & ｷｰﾏﾄﾘｸｽ ﾃﾞｰﾀ
380 DATA 19,16,12     :'ｷｰﾎﾞｰﾄﾞ ﾀﾞｲ1ﾀﾞﾝ
390 DATA 101,202,303,404,505,606,707,810
,911,000,-12,^13,¥14
400 DATA 28,33,11     :'ｷｰﾎﾞｰﾄﾞ ﾀﾞｲ2ﾀﾞﾝ
410 DATA Q46,W54,E32,R47,T51,Y56,U52,I36
,O44,P45,@15,[16
420 DATA 32,50,11     :'ｷｰﾎﾞｰﾄﾞ ﾀﾞｲ3ﾀﾞﾝ
430 DATA A26,S50,D31,F33,G34,H35,J37,K40
,L41,;17,":20",]21
440 DATA 40,67,10     :'ｷｰﾎﾞｰﾄﾞ ﾀﾞｲ4ﾀﾞﾝ
450 DATA Z57,X55,C30,V53,B27,N43,M42,",2
2",.23,/24,_25
```

【100〜130　画面初期設定】行120のPSET文はDRAW文用にロジカルオペレーションを初期化。それに続くDRAW文は、GML（グラフィック・マクロ・ランゲージ）のスケールファクタ、回転、色を初期化している
【140〜180　変数関係の初期設定】行160のユーザー定義関数は、グラフィックのLP（最終参照点）の位置をワークエリアから直接読みだすためのもの。FNP(0)がLPのX座標、FNP(2)がY座標（行260で使用）。また行170の配列X(AD, B)、Y(AD, B)は、それぞれ&HFBE5からADバイトのワークエリア（キーマトリクス）のビットBに対応するキーの左上の座標用（行330でキーの反転に使用）
【190〜280　キーボードの作画】行370以降にあるデータに読みこみ、作画。行260はそのときのLPを利用して、配列X、Yを計算設定している
【290〜350　キー入力受け付けと反応（メイン）】入力判定は、すべてキーマトリクスを調べておこなっている（行310〜320）。キー入力があると、行330でビープ音を出しながら文字の色を一瞬反転させる（フラッシュする）
【370〜450　作画データとそれぞれに対応するキーマトリクスデータ】行380、400、420、440は、キーの段ごとのデータで、それぞれ順に、段の左上X座標、Y座標、キーの数。それ以外は、各キーのデータで、3文字単位。各文字は、左から、キーに対応する文字、対応するキーマトリクスのアドレス−&HFBE5(AD)、おなじくビット(B)

BASIC ランゲージ

DRAW文13連発

BASIC自身も言語だが、そのなかにまた小さな双子の言語がある。1つが音楽用のMML、もう1つが今回のテーマ、グラフィック用のGMLだ。

グラフィック命令「DRAW」は、こんなに便利でおもしろいのに、どういうわけかそのわりに使われることが少ない。もちろん、目端のきいた投稿プログラマのなかには、DRAW文を効果的にさりげなく使っていたり、ウルトラC的な使い方をしてみせたりしている人が少なくない。しかし、一般に見渡してみると、どうもDRAW文は敬遠される傾向にあるようだ。

なぜか。それは、DRAW文を使うために、もう1つよけいに言語を覚えなくてはいけないからだろうと思う。その言語とは、DRAW文のあとに続くデータを記述するための言語で、GML(Graphic Macro Language)という。

音楽のMML(Music Macro Language)と名が似ているが、その中身も使われ方(MMLのほうはPLAY文のデータを記述する)もよく似ている。しかし、GMLのほうがずっとかんたんで覚えやすい。MMLをあやつるには、多少の楽典の知識や拍数、和音の処理など立体的な苦労がともなうが、GMLのほうはだれだって2歳のころからやっている線がきがメインテーマなので、どうやってもむずかしくなりようがない。

ただ、方眼紙のマスを数える根気は必要かもしれない。

1 上下左右

じっさいのDRAW文のサンプルを提示しながら、GMLのポイントを解説していこう。1発目は、もっともわかりやすい上下左右に線を引くコマンドである。

GMLではコマンドがアルファベット1字を基本にできている。こういう場合にありがちだが、GMLでは原則として機能を表す英単語の頭1字をとってコマンドとしている。たとえば、上はUPだからU、下はDOWNでD、左右はもちろんLとR。このコマンドに線の長さを指定する数値をくっつければ十字のGMLはリスト1の行1000のようになる。行100、110のDRAW文は項目4、項目5、項目8と関連する設定だ。

右行って、下行って、右行って……

●リスト1

```
100 COLOR 4,15,0:SCREEN 5,0:DRAW "S4A0"
110 DRAW "BM120,80"
120 READ A$:DRAW A$
130 GOTO 130
1000 DATA R10D10R10D10L10D10L10U10L10U10
R10U10
```

2 斜め4方向

「原則」をいったすぐあとで気が引けるが、斜め4方向に線を引くためのコマンドは例外的に頭文字になっていない。「右上」とか「左下」をアルファベット1字で表す適切な方法がなかったのだろう。斜めの4方向を表すGMLコマンドは、右上から左上までぐるっと時計回りにE、F、G、Hとなっていて、リスト1のプログラムの行1000をリスト2のものに置き換えれば、斜めの十字が描かれる。どうせ続きのアルファベットにするのならA～Dにすればよさそうなものだが、その4つはほかの主要コマンドが予約ずみでEからしか残っていなかったのである。

E
H
F
G

ドットが正方形でないのでちょっと歪む

●リスト2(+リスト1の行100～130)

```
1000 DATA E10F10E10F10G10F10G10H10G10H10E10H10
```

3 かかずの"B"

線をかく機能しかなければ、ひと筆書きの絵しかかけない。だから、線をかかない機能も必要だ。そのためのコマンドが、おそらくBlind(隠れた)からきたBだ。

Bが、ほかのコマンドの頭にくっつくと、線をかく仮想的なペン先(Last access Pointの略で、LPと呼ぶ)だけ移動して線を表示しなくなる。写真の矢印には、最初と最後にシッポがあるのだがBコマンドで見えなくなっている。

↑おしりに見えないシッポがある

●リスト3(+リスト1の行100〜130)

```
1000 DATA BE10F5E10F5U20L20F5G10F5BG10
```

4 回転する"A"

Angleから来たと思われるAコマンドは、以降の図形全体を90度単位で回転して表示する(パラメータ0〜3で反時計回りに指定)。

リスト1+リスト4は前項の矢印をA0(初期)〜A3でかく。

このAコマンドは一度指定するとプログラムを消しても効果は消えない。だから、リスト1の行10末尾のような配慮が必要なのだ。

↑シッポを持ってぐるんと振り回した感じ

●リスト4(+リスト1の行100〜130)

```
121 DRAW "A1":DRAW A$
122 DRAW "A2":DRAW A$
123 DRAW "A3":DRAW A$
1000 DATA BE10F5E10F5U20L20F5G10F5BG10
```

5 大きくなる"S"

GMLのマス目の単位は初期状態では1マス1ドットだが、Sコマンド(Scale)はそれを4分の1ドット単位で変更できる。項目4の図形を、1マス1ドット(初期値S4)、1.5ドット(S6)、2ドット(S8)、3ドット(S12)で表示してみた。

Sコマンドも効果があとに残るという性質はAコマンドと同様だ。

↑左上の矢印は誤差でおしりが切れている

●リスト5(+リスト1の行100〜130)

```
121 DRAW "A1S6":DRAW A$
122 DRAW "A2S8":DRAW A$
123 DRAW "A3S12":DRAW A$
1000 DATA BE10F5E10F5U20L20F5G10F5BG10
```

6 色はやっぱり"C"

GMLはそのときの描画色(COLOR文の第1パラメータやグラフィック命令で指定された色)で線をかくが、GMLのなかで色を切り換えることもできる。それがCコマンド(Color)だ。続くパラメータはカラーコード指定だ。

ところで、ほかのグラフィック命令で指定された色はDRAW文に影響するが、このCコマンドで指定した色はDRAW文でかく線にのみ効果を持つという点に注意。

↑4つとも色を変えてみた

●リスト6(+リスト1の行100〜130)

```
121 DRAW "A1S6C9":DRAW A$
122 DRAW "A2S8C7":DRAW A$
123 DRAW "A3S12C10":DRAW A$
1000 DATA BE10F5E10F5U20L20F5G10F5BG10
```

7 放射する"N"

線をかくコマンドはすべてLPと呼ばれる仮想的なペン先を移動しながら線をかくのだが、Nコマンド(たぶんNot)はこれに続く描画コマンドにかぎり線をかいてもLPを移動させない働きがある。

たとえばリスト7のように「NH5〜」などを挿入すると、矢印の頭に毛をはやすことができる。

↑15文字の追加で頭に毛が5本

●リスト7(+リスト1の行100〜130)

```
121 DRAW "A1S6C9":DRAW A$
122 DRAW "A2S8C7":DRAW A$
123 DRAW "A3S12C10":DRAW A$
1000 DATA BE10F5E10F5U20 NH5NU7NE5NR7NF5
 L20F5G10F5
```

8 位置指定の"BM"

リスト1の行110は、かきはじめの点を指定するためのものだ。使われているコマンドはすでに触れたBと次項のMの組み合わせだが、あまりによく使う組み合わせなので「BM」としてセットで覚えておいたほうがいい。このセットに続いて、座標を「〈X座標〉,〈Y座標〉」で指定する。

↑BMで好きなところに表示

●リスト8(+リスト1の行100〜130)

```
121 DRAW "BM20,150C9":DRAW A$
122 DRAW "BM120,150C7":DRAW A$
123 DRAW "BM220,150C10":DRAW A$
1000 DATA BE10F5E10F5U20 NH5NU7NE5NR7NF5
 L20F5G10F5BG10
```

9 座標指定の"M"

Mコマンドは、LPから、Mコマンドに続いて指定する座標まで線を引く。

ほかのGMLコマンドとちがって絶対座標を指定するので、どんな角度の線も自由にかける。写真のように非常にてきとうなフォルムの星だってかけるわけだ。

ただし、SコマンドとAコマンドの影響を受けない。GMLのなかにいるLINE文といった感じのコマンドだ。

実際には、「BM」のセットで使われることのほうが多いようだ。

方眼紙にかいてデータをとった

●リスト9(+リスト1の行100〜130)

```
1000 DATA "M124,130 M95,100 M140,85 M97,
130 M120,80"
```

10 コウモリ的"M±"

座標指定部分の先頭に+かーをかくと、Mコマンドは相対指定モードになる。これはきわめて重要だ。相対指定モードの場合は、移動単位で指定するので、Sコマンド、Aコマンドなどの影響を受ける。だから、前項のてきとうな星もこのモードなら移動したり拡大したり回転したりできるのだ。

いいかげんな星のバリエーション

●リスト10(+リスト1の行100〜130)

```
121 DRAW "BM40,130":DRAW A$
122 DRAW "BM80,150A1":DRAW A$
123 DRAW "BM130,150A1S8":DRAW A$
1000 DATA "M+4,50 M-29,-30 M+45,-15 M-43
,45 M+23,-50"
```

11 応用1:花

リスト11には、まだ触れてないGMLの特徴に関わる行がある。

1つは、ロジカルオペレーションの問題だ(SCREEN5以降)。GML自身にはロジカルオペレーションを操作する機能がないが、影響だけはもろに受ける。ロジカルオペレーションは、最後に指定されたモードをずっと維持するので、行110のように初期化するといい(指定省略はPSET)。

もう1つは、GMLの複合化だ。行1000のデータは、花の4分の1をかくGMLで、それをA$として読みこむ。D$はそのA$の複合で、回転してくりかえす。行130の「XA$;」はA$に入ったGMLを実行するという意味だ(71ページの表参照)。これは、複雑な、しかしパターンのある図形をかくときに威力を発揮する。

もう1つ、GML中で数値変数が使える。たとえば、D$中の「C=C;」、また行180の「BM+=X;,=Y;」。これも71ページの表で確認してほしい。プログラム中でダイナミックな使い方をする場合には欠かせない機能だ。

15色の花が次々に表示されていく

●リスト11

```
100 COLOR 15,0,0:SCREEN 5,0:DRAW "S4A0"
110 PSET (0,0),0
120 READ A$
130 D$="C=C;XA$;A1XA$;A2XA$;A3XA$;A0"
140 FOR I=1 TO 5:CLS:DRAW "BM120,110"
150  R=RND(-(I+100))
160   FOR J=0 TO 10
170    X=10-RND(1)*20
175    Y=10-RND(1)*20:C=5+J
180    DRAW"S16BM+=X;,=Y;S4"
190    DRAW D$:PAINT STEP(0,0),C,C
200   NEXT
210  TIME=0:FOR W=0 TO 60:W=TIME:NEXT
220 NEXT
230 GOTO 140
1000 DATA BU28R2F3DFD2FD5RE2R2ER2ER4D3G1
D2G1D2G2D1R5F1R2F1R1F3D3BL28
```

12 応用2:カラフル

リスト12には、リスト11に比べて語るべきポイントが少ない。

もっともいいたかったのは、SCREEN5ではCコマンドに続くパラメータの値の範囲は0〜15だが、SCREEN8では0〜255になるという点だ。あたりまえといえばあたりまえだが。

BASICから見ると、YJKモードのSCREEN10〜12もSCREEN8とおなじなので、DRAW文でも、SCREEN 8の256色やYJKの色の幅広さを活用できることになる。

そこで、YJK用のサンプルを提示できればいいのだが、諸般の事情でSCREEN8の256色サンプルだけを掲載する。

リスト12は、DATA文中においてあるGMLのなかに「=S;」や「=C;」という、行140の変数S、Cの更新とリンクする部分が組みこまれているという点以外、とくに注釈する部分はない。実行すると、カラフルな八角形をえんえんと塗りなおしはじめる。Sコマンド、Cコマンドを使えばちょっとしたGMLアニメができるというサンプルでもある。ほかにもアニメの可能性は考えられる。

色は内側から変化していく

●リスト12

```
100 COLOR 15,0,0:SCREEN 8,0:DRAW "S4A0"
110 PSET (0,0),0:F=1
120 DRAW"BM20,80"
130 READ A$
140 S=S+1:C=C+F:IF S>255 THEN S=1:F=-F
150 DRAW A$
160 GOTO 140
1000 DATA S=S;C=C;ERFDGLHU
```

13 応用3：2人の漱石

AVフォーラムの『千円札』(おめあて作／1990年5月号にはじめて掲載され、1991年10月情報号で再掲)の夏目漱石GMLは、ぼくの知るかぎり、GMLの最高傑作だと思う。この夏目漱石のGML部分をかりて、ちょっとした文字列操作をやってみた。

行1000から1020が夏目漱石をかくGMLデータだ。このGML中のLをR、RをL、斜めのEFGHをそれぞれ左右反対のものに置き換えてみると夏目漱石が左右反対になる。行160でGMLから1字ずつ取り出し、行170で置き換えるべき文字かどうかをチェック、行180で置き換える処理をやっている。

GMLは、基本的に文字列なので、文字列操作のテクニックをそのまま適用して、さまざまなバリエーションで遊べるのだ。

←左側をかいてから右側をかく

●リスト13

```
100 CLEAR 1000:DEFINT A-Z:DIM A$(2)
110 COLOR 15,4,4:SCREEN5:COLOR=(15,2,0,4
):COLOR=(4,7,7,7):DRAW"S4A0"
120 DRAW "BM100,130"
130 FOR I=0 TO 2
140  READ A$:DRAW A$:L=LEN(A$)
150   FOR J=1 TO L
160    CH$=MID$(A$,J,1)
170    CH=INSTR("RLEFGH",CH$)
180    IF CH THEN MID$(A$,J,1)=MID$("LRH
GFE",CH,1)
190   NEXT:A$(I)=A$
200 NEXT
210 X=90:Y=195:PAINT(X,125):PAINT(X,170)
:PAINT(75,Y):PAINT(110,Y)
220 DRAW "BM150,130"
230 FOR I=0 TO 2:DRAW A$(I):NEXT
240 X=160:Y=195:PAINT(X,125):PAINT(X,170
):PAINT(175,Y):PAINT(140,Y)
250 GOTO 250
1000 DATA L4HL3GLHL7GL5GL2GDGDGD4GD6GD4G
D2U13HU3EU2EU2EU3EU2REUER3E2R3EURFRERER1
0FRFRF5R2F2DFDF2D2FD6GDFD3GFRFGLGLR2DRFD
U3NLENUFD4GD3GD3GD3G2DL2H2EUBU2UR2URU3RU
4LULDLDGLD4GD4U4EU3RE2DFLD3LDBD13R2FDFDF
5D22L27UEHG2DL8U2EUEUH3UD6
1010 DATA GD3BU11NG2U2HU3GLG2LGLGLGL2GLG
L2GLD8R16U2EU5BR6FER8ERE3UEUE3UEUE3UE4UH
U5HD7LG5LGL2UHUDFDFGHDLFLDL2DNL3GNF2L8GR
D2FDFD2BU8L7HGUHUL2ULHUH3U3HU3HU7HU4LU3R
D2U5EU2RUEURUEDR8D3FDFD2GDBU8BLL2EL7G2BD
4R2UEFURUR2GR2FL3FL2BD5GLHBD13FD
1020 DATA BR2EUERE2REHLBL2UBD3BR4R2E2BU3
URBD4DBE3FDGBL2GFR3F5D2GL2HL3BF2L3DL6HBH
3DL3U2FR14BR9U4BU3UHUBR5BU2EUHUEBUBEUHBL
2BHL2GHGL2HLHLER2URF2DL2U2R3DRDBD5NELGL2
HBE2L2HBL3FH2GU2EBU4R2UR6DR3DRFUH2L2ULHL
4GNR6LGRBR14EHBH3BUU2BU5HBU4E2FG2DF2D4FD
4F3DFDFD3FU2E3
```

GML(グラフィックマクロランゲージ)一覧表

コマンド	書式	コマンドの意味と使用例
移動コマンド(方向・距離指定)		
U	Un 上にn単位移動	●指定された方向(8方向)へ線をかいていくコマンド。方向は下図参照。それぞれのコマンドに続く移動単位の指定(各コマンドの直後につける数値。左の書式ではnで表記)を省略するとn＝1とみなされる。移動単位の大きさは、右の段のSコマンドによって変わるが、初期状態(S4)では、1単位＝1ドットになっていて、たとえば「DRAW "U4E4R4F4D4G4L4H4"」とすれば、上に4ドット移動しながら線をかき、そこから右上へ縦横4ドットぶん移動しながら線をかき……というふうにして八角形を描くことになる。ただし、始点の1ドットは数えないので、nドット移動しながら線をかくと、最終的にn＋1ドットぶんの線をかくことになるという点に注意。
D	Dn 下にn単位移動	
R	Rn 右にn単位移動	
L	Ln 左にn単位移動	
E	En 右斜め上にn単位移動	
F	Fn 右斜め下にn単位移動	
G	Gn 左斜め下にn単位移動	
H	Hn 左斜め上にn単位移動	
移動コマンド(座標指定)		
M	Mx,y 座標(x,y)まで線をかく	●Mの直後に指定される座標まで線をかく。たとえば、「DRAW "M100,100"」ならLPから座標(100,100)まで線をかく。座標指定なのでA、Sコマンドの影響はない。
	M±x,y X方向にX単位、Y方向にY単位移動して線をかく	●Mの直後に＋かーの符号があるとX軸方向にx単位、Y軸方向にy単位移動する相対指定モードになる。「DRAW "M+10,10"」ならLPからX方向に10単位、Y方向に10単位移動する。この場合はA、Sコマンドが影響する。

コマンド	書式	コマンドの意味と使用例
修飾コマンド(移動方法の指定)		
B	B(移動コマンド) 移動するが描画しない	●Bコマンドのついた移動コマンドは、LPは移動させるが、線をかかない。「DRAW "BM50,50"」なら座標(50,50)にLPが移動する。座標指定によく使われる。
N	N(移動コマンド) 移動するがLPはもとのまま	●Nコマンドのついた移動コマンドは、線はかくが、LPを移動させない。「DRAW "NLNUNR"」なら1点を中心に左、上、右に線をかき、LPはもとのまま。
特殊コマンド(倍率、回転、色指定)		
S	Sn 1単位をn/4ドットに指定	●移動コマンドで指定する移動単位のドット数を指定する。Sの直後に置かれる数(書式ではnで表記)の4分の1が、1単位のドット数となる。「DRAW "S8U2"」なら上方向に4ドット移動する。初期状態はS4。
A	An n×90度左に回転	●あるGMLでかかれる図形を90度単位で反時計回りに回転して表示する。「DRAW "A1U2"」なら左に2単位移動。
C	Cn カラーコードnで描画	●DRAW文でかく線の色をカラーコード(パレットコード)で指定する。「DRAW "C1"」なら黒を指定。
間接指定モード(GML中で変数を使う)		
X;	X(文字変数); 文字変数の内容をGMLとして描画	●文字変数をGMLの一部として組みこむ。前後にほかのGMLがあってよい。たとえば「A$="URDL"」としてあれば「DRAW "S16XA$;"」は「DRAW "S16URDL"」とおなじで縦横5ドットの四角をかく。
=;	＝(数値変数); 数値変数の内容をGML中の数値とする	●数値変数をGMLの一部として組みこむ。たとえば「A＝16」としてあれば「DRAW "S=A;URDL"」は「DRAW "S16URDL"」とおなじことになる。

BASICピクニック

大きな桁の計算へのアプローチ

そのままでは扱えない100桁くらいの数値で足し算、引き算、かけ算、割り算をするにはどうすればいいのか。そのための基本的な考え方とプログラムの組み方に接近してみた。

MSX FAN 付録ディスク
→P108の「スーパー付録ディスクの使い方」参照

考えてみると、いきなり古い話を持ち出すようで気が引けるが、1990年12月号のBASICピクニックで「#18電子計算機としてのMSX」をやったとき、πを500桁まで計算するプログラムを掲載した。このときの基本テーマは、「科学技術計算」だったので、おもにπの値を計算する方法についての話に終始した。

しかし、πを500桁まで計算するにあたっては、1つの大きな難関をクリアする必要があった。それは、ふつうの変数に大きな桁数の数値を代入すると、一定桁数以上は無視されてしまうという点だ。たとえば、

10 P=3.141592653589793

という行を試しに打ちこんでみよう。すぐにリストを見ると、

10 P=3.1415926535898#

とかってに変えられてしまう。

基本的にMSXでは、最大でも15桁目を四捨五入した14桁までの数値しか扱えないのだ。

前述のπを500桁まで計算するプログラムでは、そこをプログラム上のくふうでなんとかしているわけだが、その「くふう」の部分を今回のメインテーマとして掘り下げつつ、いくつかのサンプルを示そう。

接近1 MSXが扱う3つのタイプの数値とは

最初に、MSXでは限られた桁数の数値しか扱えないという点について明確にしておこう。

これはどんなBASICマニュアルにもかならず書いてあることだが、MSXの「数値」には、以下の3つのタイプしかない。

①−32768〜32767の整数
②有効数字6桁の実数　※1
③有効数字14桁の実数

①はともかく、有効数字とか実数とか、日常的にはめったに使わないことばなのでちょっとわかりにくいかと思うが、②を翻訳するとまあだいたい「意味のある桁が6桁以内の、少数点付きのものを含む数値」ということになる。

意味のある桁とは、前や後に続く0以外の桁ということで、たとえば、12300000や0.00123では3桁(「123」の部分)、12030000では4桁(「1203」の部分)となる。

それぞれ、①は「整数型」、②は「単精度実数型」、③は「倍精度実数型」と呼ばれている。わざわざ「型」という字をつけるのは、実世界でいう「整数」や「実数」とは厳密にはちがうからだ。

MSXが数値を記憶したり、計算したりするときは、この3つの型のうちのどれかになっている。当然だが、これに対応して数値変数にも3つの型があり、とくにことわらないかぎり、変数はすべて倍精度実数型になっている。

それぞれの型の許容範囲を超えるときは、整数型では小数点以下切り捨て、2つの実数型では範囲を超える桁で四捨五入した値をその数として扱う。

そうすると、BASIC上の数値として処理しているかぎりは、15桁以上の桁の数は無効だということになる。

この3つの型の呪縛から逃れる1つの方法に、文字を使う方法がある。数値としては、最大有効数字が14桁までしか扱えないが、文字としてなら、最大255桁までを正確に扱うことができる。

もちろん、文字のままでは計算することもできないが、大きな桁数を扱う場合、あるプロセスで数を文字として扱う必要が出てくるだろう。そういう意味で、文字としての数は、数の第4の型のようなものといえるかもしれない。

接近2 n進数という考え方

大きな桁の数を計算する方法を考えるまえに、まず計算のときにどういうことが問題になるのかを確認しておく必要がある。

たとえば、ふつうの計算でも、123+234などはかんたんに答えが出せるが、987+678などになるとちょっとつまってしまう。この2つの計算の違いは、繰り上がりがあるかどうかだ。同様に、543−321はかんたんだが、321−234もちょっとつらい(これは繰り下がり)。かけ算や割り算は、足し算、引き算を複合した計算になるので、けっきょく、計算のむずかしさは、繰り上がり、繰り下がりにあるといえるだろう。

繰り上がり、繰り下がりは、1桁の数に左右される。日常生活では、1桁は10の何乗と決まっている。1の位は10の0乗、10の位は1乗、100の位は2乗……。こういう数の管理法のことを、10進法といい、そういう管理をされている数を10進数という。

これとまったく同様に、2進数では2の何乗かが1桁の数になり、16進数では16の何乗かが1桁の数となる。

10進数で計算しているときにはあまりピンとこないのに16進法で計算してみるとよくわかるのは、けっきょく、計算は桁ごとの単純な計算と繰り上がりの組み合わせだということだ。それは2進数でも10進数でも16進数でも、100進数でも200進数でもおなじ。1万進数でも、千万進数でもおなじだ。ここに、大きな桁の計算のための、1つのヒントがある。

表1 2進数/10進数/16進数

2進数	10進数	16進数
(&B) 1	1	(&H) 1
10	2	2
11	3	3
100	4	4
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10

接近3 配列を使って千万進数をシミュレートする

以上の2つの話を総合して、とりあえず足し算を実現してみた。

まず、2つの数の入力は、てっとりばやく文字で受け入れることにした。これなら255桁まで受け入れられる。もしそれ以上の桁数が必要なときは、複数の文字変数に分割して入力すればいい。もっとも、人間が200桁以上の数値をその場で入力するというのは、現実的な話ではないが。

さて、そうして受け入れた大きな桁の数を、どうやって計算するのか。接近2で述べた、計算の本質は繰り上がりにあるという洞察がここで生きてくる。けっきょく、倍精度実数型の限界は、この繰り上がりが14桁の範囲内でしかできないということなのだ。

では、できない部分をプログラムで補えばいい。繰り上がりをともなって計算できる部分はふつうに計算し、繰り上がりの限界に達するまえに、別枠の計算システムに繰り上がり情報を移せばいい。そして、その上の桁の計算は、それまでの桁とは別個の形で新たにおこなう。そうすれば、有効数字を14桁以内に抑えたまま、繰り返し繰り返し上の桁を計算していくことができる。

それを実現するのが、2次元配列Aだ。A(0, n)とA(1, n)に2つの数を入れ、A(2, n)に答えを入れている。

配列の1つの要素には、7桁ずつを入れている。7桁なら、要素どうしをかけ算しても結果が14桁を超えないからだ(足し算だけなら13桁でもいいはずだが)。

2つの数の足し算は、対応する配列どうしを下位から足し算して、それをA(2, n)に保存していくという形で実現している。このときに結果が7桁を超えると、変数Eを1にして、次の配列要素の足し算に含めている。これは繰り上がりのシミュレートで、この計算全体は千万進数をシミュレートしているというわけだ。

●図1 配列に大きな数を収める

3431162744734833350965

……	3	4311627	4473483	3350965
……	A(0, 3)	A(0, 2)	A(0, 1)	A(0, 0)

●図2 配列を使った足し算の仕組み

4473483 A(0, 1) ← 1
3350965 A(0, 0) + 8946326 A(1, 0) → 12297291
5526516 A(1, 1)
2297291 A(2, 0)

●リスト1 大きな桁数の足し算

```
100 SCREEN 0:WIDTH 40:KEYOFF:CLEAR 1000
110 N=300:F=7:H=N¥F:U=10^F
120 DIM A(2,H),A$(1),LA(1)
200 '*** input
210 PRINT "Input 2 numbers"
220 FOR I=0 TO 1:PRINT "number";I+1
     :LINEINPUT A$(I):NEXT
230 FOR I=0 TO 1
240   Z    =(F-LEN(A$(I)) MOD F) MOD F
250   A$(I)=STRING$(Z,"0")+A$(I)
260   LA(I)=LEN(A$(I))¥F-1
270   FOR J=0 TO LA(I)
280     A$=MID$(A$(I),1+J*F,F)
290     A(I,LA(I)-J)=VAL(A$)
300   NEXT
310 NEXT
1000 '*** tashizan
1010 IF LA(0)<LA(1) THEN LA=LA(1)
                    ELSE LA=LA(0)
1020 FOR I=0 TO LA
1030   A(2,I)=A(0,I)+A(1,I)+E
1040   IF A(2,I)>=U THEN A(2,I)=A(2,I)-U
        :E=1        ELSE E=0
1050 NEXT
1060 IF E THEN LA=LA+1:A(2,LA)=1
2000 '*** answer
2010 PRINT:PRINT "ANSWER"
2020 IF A(2,LA)=0 AND LA>0
      THEN LA=LA-1:GOTO 2020
2030 PRINT MID$(STR$(A(2,LA)),2);
2040 IF LA THEN LA=LA-1 ELSE END
2050 FOR I=LA TO 0 STEP -1
2060   PRINT RIGHT$(STR$(A(2,I)+U),F);
2070 NEXT:END
```

接近4 引き算は足し算よりちょっとむずかしい

引き算を考えるときに、もっとも大きな問題は、もしかしたら答えがマイナスになるかもしれないということだ。2つの道がある。

1つは、あらかじめ2つの数の大小を比較して、大から小を引く、つまり中間結果をかならずプラスにするというやり方だ。そのあとで、符号をつけたりつけなかったりすればいいわけだ(わたしたちの日常の引き算はこのシステム)。

もう1つは、右のリスト2(行310まではリスト1を使用)で使っている、補数の概念を使ったやり方だ。たとえば、10の位を無視すれば、5－1という引き算は5＋9で代用できる。この1に対する9のような数を補数という。

●図3　配列を使った引き算の仕組み

4473483 ← -1
A(0, 1)
5526516
A(1, 1)
-5595361
+10000000
4404639
A(2, 0)
－
3350965
A(0, 0)
8946326
A(1, 0)

●リスト2　引き算のための主要ルーチン

```
1000 '*** hikizan
1010 IF LA(0)<LA(1) THEN LA=LA(1)
                    ELSE LA=LA(0)
1020 FOR I=0 TO LA
1030   A(2,I)=A(0,I)-A(1,I)+E
1040   IF A(2,I)<0 THEN A(2,I)=U+A(2,I)
        :E=-1    ELSE E=0
1050 NEXT
1060 IF E THEN FOR I=0 TO LA
             :A(0,I)=0:A(1,I)=A(2,I)
             :NEXT:LA=LA+1:A(0,LA)=1
             :SG=-1:E=0:GOTO 1020
2000 '*** answer
2010 PRINT:PRINT "ANSWER"
2015 PRINT LEFT$(STR$(SGN(SG)),1);
2020 IF A(2,LA)=0 AND LA>0
       THEN LA=LA-1:GOTO 2020
2030 PRINT MID$(STR$(A(2,LA)),2);
2040 IF LA THEN LA=LA-1 ELSE END
2050 FOR I=LA TO 0 STEP -1
2060   PRINT RIGHT$(STR$(A(2,I)+U),F);
2070 NEXT:END
```

接近5 かけ算は筆算の要領で

じつは、かけ算はある考え方によれば引き算より楽にプログラムできる。その考え方とは「A×BとはAをB回足すことだ」という、きわめてノーマルな考え方だ。これに基づけば、リスト1を複数ループになるように書き換えればいいだけだ。

しかし、これは実用的でない。たとえば、1兆×1兆は、1秭(じょ)(10の24乗)という数になるが、もし1兆を1兆回足すとなると、ターボRでも、赤ちゃんが生まれてハイハイするくらいまでの時間はかかると思う。

やはり、かけ算は「*」を使って計算したい。となると、筆算システムを利用するのがいい。

たとえば、12×34なら、4×2＋4×10＋30×2＋30×10というふうに各桁ごとにかけ合わせたものを合算すればいい。ここでは、1配列を1桁とする千万進数なので、桁ごとのかけ算を配列ごとのかけ算に置き換えればあとは桁上がりのくふうでなんとかなる。リスト3は、行100～310と行2000以降はリスト1と共通。

●リスト3　かけ算のための主要ルーチン

```
1000 '*** kakezan
1005 DIM E(H) '**ｼｮｷｾｯﾃｲ ﾆ ﾂｲｶ
1010 LA=LA(0)+LA(1)+1
1020 FOR I=0 TO LA(1)
1030   FOR J=0 TO LA(0):K=I+J
1040     A=A(0,J)*A(1,I)
1050     A(2,K)=A(2,K)+A+E(K):E(K)=0
1060     IF A(2,K)>=U
           THEN E=INT(A(2,K)/U)
               :E(K+1)=E(K+1)+E
               :A(2,K)=A(2,K)-E*U
1070   NEXT
1080 NEXT
1090 A(2,LA)=E(LA)
```

接近6 割り算の特殊性

引き算もかけ算も、なんのかんのといいながら、けっきょくは足し算の変形だった。ようするに、＋、－、×という計算は、構造的に単純なわけだ。

ところが、割り算は異質で、今回の記事のボリュームでは割り算までをカバーすることができない。

ここでは、割り算の特殊性と割り算を考える最初の第1歩について軽く触れるていどにする。

まず、すぐに気がつくのが、小数点の登場だ。しかし、これはじつはたいした問題ではない。小数点はたんに桁の目印にすぎず、3桁区切りの「,」と本質的におなじなのだ。千万進数としては、答えを表示するときに、どの配列の直後にピリオドを表示すればいいかをなんらかの形で記憶しておけばいいわけだ。

だが、答えの精度があがるたびに数が下位の桁のほうへ伸びていくという点は問題だ。これまで、上位の桁へと答えが伸びていった(もしくはそれ以内に収まった)のとは逆で、このことは配列での数値の持ち方に直接影響してくる。これまでとおなじ構造の配列をむりやり使うとすると、新しく計算結果の精度があがるたびに、それまでの配列の内容をぜんぶ上位の配列へ移して、最下位の配列に新しい成果を入れる……というやり方をせざるをえなくなる。

これは、配列の数値の持ち方を図4のようにすればいい。

もう1つの問題は、割り算全体のシミュレートだ。筆算の割り算は、答えの見当をつけ、かけ算と引き算で確認し、答えの桁を1つ1つ決定していく。配列でも同様のシステムになるだろう。

●図4　割り算のときの数の収め方

3431162|7447348|3335096|5

3431162 A(0, 0)
7447348 A(0, 1)
3335096 A(0, 2)
5 A(0, 3)
…………

総合的な実用例「πを500桁まで計算するプログラム」

大きな桁の数の加減と割り算の実用例

●リスト4　PAI.BP2(このプログラムのみディスクに収録)

```
100 COLOR 15,0:SCREEN 0:WIDTH 40:KEYOFF
110 CLEAR 200:DEFINT A-C,E-N
120 N=500:M=5:NM=(N-1)¥M+1:DIM M(1,NM)
130 X=2/LOG(10):Y=LOG(5)*X:Z=LOG(239)*X
140 FOR I=1 TO NM
      :M(0,I)=M*I/Y+1.5
      :M(1,I)=M*I/Z+1.5:NEXT
150 H=N¥9+1:U=10^9
160 DIM S(1,H+1),T(1,H),P(H),D(1)
    :T(0,0)=4*4*5:D(0)=5^2
    :T(1,0)=4*239:D(1)=239^2
170 FOR I=1 TO NM
180   FOR J=0 TO 1
190     FOR K=M(J,I-1)+1 TO M(J,I)
200       FOR A=0 TO H
            :S(J,A)=T(J,A)
          :NEXT
210       D=D(J):GOSUB 1000
220       FOR A=0 TO H
            :T(J,A)=S(J,A)
          :NEXT
230       D=2*K-1:GOSUB 1000
240       IF (K+J)MOD2
            THEN F=1
            ELSE F=-1
250       GOSUB 2000
260     NEXT
270   NEXT
280   C=0:G=0:LOCATE 0,0
290   PRINT "π =";STR$(P(0));".";
300   G=G+1:A=0
310   A$=RIGHT$(STR$(P(G)+U),9)
320   A=A+1:C=C+1:PRINT MID$(A$,A,1);
330   IF C MOD 5  =0
        THEN PRINT " ";
340   IF C MOD 25 =0
        THEN PRINT:PRINT SPC(6);
350   IF C=N THEN END
360   IF C=M*I
    THEN NEXT
370   IF A=9 THEN 300 ELSE 320
1000 '●●● WARIZAN SUB
1010 FOR A=0 TO H
1020   Q=INT(S(J,A)/D)
1030   R=S(J,A)-Q*D
1040   S(J,A)=Q
1050   S(J,A+1)=S(J,A+1)+R*U
1060 NEXT:RETURN
2000 '●●● KAGEN SUB
2010 FOR A=H TO 0 STEP -1
2020   P(A)=P(A)+(S(J,A)+E)*F:E=0
2030   IF P(A)>=U OR P(A)<0
         THEN E=1:P(A)=P(A)-U*F
2040 NEXT:RETURN
```

100〜110　画面初期化／文字列領域の初期化／倍精度でなくてはならない変数 以外は、すべて整数型とする

120　最終結果の小数点以下の桁数(N)、結果表示のときの区切り桁数(M)、最終結果はM桁のブロック何個ぶん(NM)かの変数初期化／配列M(n,m)の宣言⇒πを求めるマチンの公式は2つの無限級数(分母が5系統のものをS0、239系統のものをS1とする)の差の形。答えの第mブロックまでの精度を確実にするにはSnの何項までの計算が必要かを表す配列

130〜140　配列M(n, m)の値の計算・設定

150　答え用配列の大きさH、U進数(ここでは10億進数)のUを設定

160　Sn計算用配列S(n,m)、配列S(n,m)の一時保存用配列T(n,m)、答えを入れる配列P(n,m)、無限級数Snの割り数D(n)の初期化

170　第Iブロック(5桁単位)の計算開始

180　マチンの公式の無限級数SJの計算開始

190　第IブロックЕを計算するために必要な無限級数SJの第K項の計算開始

200　前回の計算結果の配列Tから配列Sへ複写

210　割り数Dの設定(5^2か239^2)／行1000(割り算サブ)を呼ぶ

220　割り算サブで値を更新された配列Sの内容を配列Tに保存

230　割り数Dの設定(分母の奇数の係数)／行1000(割り算サブ)を呼ぶ

240　計算した項目が正か負かを判定し、正負用変数Fを設定

250　行2000(足し算／引き算サブ)を呼ぶ

260〜270　それぞれ行180、180のループ閉じ

280〜370　答えの表示

280　表示桁C(小数点以下)、表示用配列番号Gの初期化／カーソルをホーム ポジションに

290　答えの最初の部分表示(「π　=　3.」)

300　G更新／配列のうち第何桁を表示するかの一時変数Aの初期化

310　答えの配列から9桁の文字列を取り出してA$に入れる

320　A更新／C更新／A$の第A桁目を表示

330　いま表示した数字が5桁ブロック最後の文字なら、次に空白を表示

340　いま表示した数字が1行の終わりの文字なら、次の空白6個を表示

350　目的の桁数まで表示したら終了

360　行170からにはじまるIのループの目的を達成したら、次のループ

370　答えの配列から取り出す数字が9桁目に達していたら、行300へ。そうでなければ行320へ

1000　割り算サブ

1010　割り算開始

1020　S(J, A)をDで割った商Qの計算

1030　おなじくあまりRを計算

1040　S(J, A)に商を入れる

1050　S(J, A+1)に余りR×Uを設定

1060　ループ閉じ／呼び出し先へもどる

2000　足し算・引き算サブ

2010　加減のループ開始

2020　答えの配列P(A)に無限級数の配列S(J,A)を加える(符号用変数Fにより、Fがマイナスのときは実質的に引き算となる)。Eは繰り上がり、または繰り下がり。

2030　繰り上がり・繰り下がり用変数Eの設定

2040　ループ閉じ／呼び出し先へもどる

BASICテクニック

鈴木智博氏の10の質問

このページへの読者からの質問は、一部分ずつさりげなく取り入れさせていただいているが、今回は1人の読者が連射してきた10の質問に1つ1つ答えてみよう。

MSXFAN 付録ディスク
➔P108の「スーパー付録ディスクの使い方」参照

愛知県の鈴木智博という読者(年齢不詳)から、1枚の質問ハガキが届いた。質問ハガキそのものはとくに珍しくはないし、「バボさんってほんとうはすごいブスなんですか」というようなよくわからない質問以外は、できるだけ記事に取り入れながら作っているので、ハガキそのものを誌面に出すことはめったにない。だが、鈴木氏の質問はなんだか鬼気迫るものがあった。

このハガキは、もともとスーパービギナーズ(超初心者)講座あてのおしかりのハガキだった。超初心者と銘打っておきながら、むずかしすぎる、「正真正銘の超初心者」である「僕」には"わかんねえ"の対象」だ、という趣旨の前文があり、続いて質問を書き連ねて、「説明書(マニュアル)ではわからない」のでちゃんと教えてほしい、と訴えている。

その質問は、みごとに重要かつ基本的な事項ばかりをおさえ、おそらく多くの「超初心者」にとっても聞きたかったことではないかという感触があった。

スーパービギナーズ講座にかわって、鈴木氏とその仲間たち(つまり、もしかしたらあなた)の質問に体あたりしてみよう。

Q1 ワークエリアの説明でFFFFとか書いてあるけど、数字にすると何になるのですか。また、ユーザーエリアとかの意味もよくわかりません。

A1 それは数字です。

ＦＦＦＦとか書いてあるのは、16進数の数字です。たぶん、ききたいのは「10進数にすると、いくつになるのか」ということだろうとは思いますが、もし、このアルファベットを記号の一種だと思っていたら、ちょっと困ります。

16進数は、0～9とA～Fのアルファベットを使って書き表され、たとえば31に相当する16進数は、

１Ｆ

と書かれます。ただし、BASICのプログラム中では「&Ｈ１Ｆ」というふうに頭に&Ｈを付け、Simple ASMなどのアセンブラのプログラム中では「１ＦＨ」というふうに末尾にＨを付ける決まりになっています。それ以外の文章中でも、区別がつきやすいように&ＨやＨを付けることも多いのですが、メモリのアドレスごとに説明していくワークエリアの解説などでは煩雑(はんざつ)になるので、&Ｈは省略しているのです。

●図1 16進数と10進数の変換のしかた

PRINT &H××××	××××=16進数
PRINT HEX$(n)	n=10進数

16進数を10進数に変換したり、10進数を16進数に変換したりするのはけっこうかんたんです。

PRINT &H1F

を直接実行すれば「31」と10進数に変換されて表示され、

PRINT HEX$(31)

を実行すれば「１Ｆ」と16進数に変換されて表示されます(図１)。

ただし、整数型の特性のため、16進数の８０００以降はマイナスの数値に変換されてしまいます。８０００以降を変換するときは、

PRINT &H8000+65536

というふうに、すべての答えに65536を加えてください。

さて、メモリは0～ＦＦＦＦまでのアドレスが付いていて、ふつう、0から７ＦＦＦはROM、８０００からはRAMになっています。このRAMのうち、BASIC用のワークエリアやディスクドライブ用のワークエリアに使われる部分を除いた部分が、ユーザーが自由に使っていい部分(プログラムなどを入れる)で、これがユーザーエリアです(図２)。

●図2 標準的なメモリマップ

領域	アドレス
ROM	0000
ユーザーエリア	8000
※機種によってアドレスは異なる	
ディスクのワークエリア	
ワークエリア	F380
	FFFF

Q2 VRAMって何ですか。

A2 画像用のメモリです。

VRAMは、VideoRAMの略で、画像用のメモリを意味しています。

たとえば、SCREEN 1の画面は、32字×24行の文字が表示できるようになっていますが、その1マスごとに1バイトのメモリが対応していて、そのマスに表示される文字のキャラクタコードを記憶しています。そういうメモリの集まったところを、パターン名称テーブルといいます(図3)。

キャラクタコードごとに文字の形を記憶しているのがパターンジェネレータテーブルという部分です。キャラクタコード1つにつき、8バイトを割り当てて、1文字1文字の形を記憶しています。

また、スプライトの形の情報はスプライトパターンジェネレータテーブル、スプライトの表示位置や色などはスプライトアトリビュートテーブル、文字色などの情報はカラーテーブル、パレット情報はパレットテーブルにそれぞれ記憶しています。

LSI

●図3 SCREEN1におけるVRAMとディスプレイの関係

Ø0ØØ
パターンジェネレータテーブル
Ø8ØØ
パターン名称(ネーム)テーブル
スプライトアトリビュートテーブル
2ØØØ
カラーテーブル　パレットテーブル
28ØØ
スプライトパターンテーブル
ディスプレイ

これらの情報がSCREEN1の画面情報のすべてです。

SCREEN1のプログラムを走らせると画面が刻々と変化したりしますが、それは上で紹介したVRAMの各部分(テーブル)の記憶内容が刻々と変化しているということとまったくおなじです。

VRAMとは、このように「画面」そのものなのです。

Q3 論理演算子って何ですか。COPYで使おうと思ったけど、わけがわからんかった。

A3 論理演算を示す記号のことです。

論理演算とは、数値を2進数に変換したうえで各桁ごとに1か0か計算していくものです。

COPY、LINE、PSETなどでの論理演算とは、もともとの点のカラーコードとそこに表示しようとする点のカラーコードとを論理演算で計算して、最終的に表示されるカラーコードを決めているのです。

「11　AND　6」は2になりますが、カラーコード11の四角のうえに論理演算子「AND」を使ってカラーコード6の四角を重ねると、その重なり部分はカラーコード2の四角になります(図4)。

●図4 カラーコードの論理演算

2=&BØØ1Ø
11=&B1Ø11
6=&BØ11Ø

↑リスト1の実行画面(図4とおなじ)

●リスト1　LINEの論理演算子

```
100 SCREEN 5:COLOR 1,15,0:CLS
110 LINE (50,50)-(150,120),11,BF
120 LINE (100,85)-(220,170),6,BF,AND
130 GOTO 130
```

【解説】●行100=画面初期設定。背景色を15(白)にしているのは、行120で描くパネルの背景部分に描かれる色が完全に表示されるようにするため●行110=カラーコード11のパネルを描く●行120=カラーコード6のパネルをずらしてANDで重ねる。色は、〈元にあった色〉AND6で得られる色になる

Q4 文字フォントを変えるとかあったけど、字がどうなるの。

A4 文字の形が変わるのです。

「フォント」(font)とは、もともとおなじ種類の活字セットのことで、ようするに「字体」を意味することばです。パソコンでも一般に文字の形のデータをフォントデータといいます。

SCREEN1で文字フォントを変えるということは、具体的にいうとパターンジェネレータテーブル(前項参照)を書き換えるということです。このテーブルのデータが変わると、画面に表示されている文字の形が変わります。

たとえば、SCREEN1では、文字「A」(キャラクタコード65)のパターンデータが、VRAMの&H2Ø8～&H2ØFに、図5のような数値(ただし、2進数)で収められています。

これは0を背景、1をドットとして見ると、まさに文字「A」の形になっています。これはほかの文字でもおなじです。

ここのデータを図5の矢印の先にあるデータに変更したとすると、文字「A」の形はそのデータで1がならんで作る太い「A」の形に変わってしまうのです。

データの1のならび方をくふうして、文字「A」を三角形にしたり、レンガパターンにしたりすることもできます。複数の文字を組み合わせて1つの絵になるようにもできます。じっさい、多くのゲームプログラムはそうやってキャラクタや背景を作っているのです。

●図5 文字「A」のフォントを変える

アドレス	変更前		変更後
&H2Ø8	ØØ1ØØØØØ	→	ØØ11ØØØØ
&H2Ø9	Ø1Ø1ØØØØ		Ø1111ØØØ
&H2ØA	1ØØØ1ØØØ		11ØØ11ØØ
&H2ØB	1ØØØ1ØØØ		11ØØ11ØØ
&H2ØC	11111ØØØ		111111ØØ
&H2ØD	1ØØØ1ØØØ		11ØØ11ØØ
&H2ØE	1ØØØ1ØØØ		11ØØ11ØØ
&H2ØF	ØØØØØØØØ		ØØØØØØØØ

Q5 領域の意味もよくわかりません。

A5 メモリの区画のことです。

「領域」ということばそのものは「領有している区域」「領地の範囲」(ようするに、縄張り)といった意味で、プログラム関係で出てきた場合は「メモリのある範囲、区間」といった意味で使われています。それ以上の意味はとくにありません。このことばは、じつは「エリア」という単語を日本語にしたときの訳語なのです。たとえば、ワークエリアを日本語によると「作業領域」になります。

「領域」は、開始アドレスと終了アドレスで範囲を表し、その範囲内のメモリをどういう目的に使っているかを伝えるときなどに使われます。たとえば、マシン語プログラムを収めている部分をマシン語領域と呼んだりするわけです。

Q6 キーバッファクリアってどういうことですか。

A6 たまっているキー入力情報を掃除することです。

文字キーを押すと画面にそのキーに対応した文字が表示されますが、キーを押した情報が直接画面(VRAM)に行っているわけではありません。そこへ行くまえにキーバッファというワークエリアの一種(図6)に、入力された文字のキャラクタコードをいったん順々に収めていき、そこの情報を処理していくという手順をとっているのです。入力された情報がキーバッファの最後の番地(&HFC17)までくると、また最初(&HFBFØ)にもどるようになっているので、ふつうキーバッファは自分の尻尾を飲みこむへびのようなイメージに描かれます。

さて、キー入力があっても、いろいろな理由で処理されないままの情報が残ることがあります。まさに、そういう情報もきちんと処理するためにキーバッファというものがあるのですが、プログラムが走っているときなどは、ちょっとしたことで不要な入力がキーバッファに残ってしまうこともあります。処理を再開するとその不要なキー入力を一挙に処理して、カーソルを変な位置に動かしたり、かってにリプレイしてしまったり、突然文字を表示したりすることがあり、必要に応じて、不要なキーバッファの情報を掃除することがあります。これがキーバッファクリアということです。

掃除するといっても、キーバッファの内容をゼロにするのではなく、キーバッファクリアをする時点までの未処理情報を処理済みとして扱うようにするだけですが。

◉図6 キーバッファ

アドレス	内容
&HFBFØ	A
&HFBF1	B
&HFBF2	C
&HFBF3	D
&HFBF4	↓
&HFC15	←
&HFC16	↑
&HFC17	DEL
	RETURN

40バイト

Q7 エスケープシーケンスって何ですか。

A7 エスケープコードと特定の文字の組み合わせのことで、特別な機能を持っています。

「エスケープ」とは、本来「逃亡」などの意味を持つことばですが、コンピュータではおもに「拡張」という意味で使われます。

エスケープコードはESCと略記され、CHR$(27)です。

このエスケープコードに続けて、大文字のYやL、Mなど、いくつかの特定の文字を続けてPRINT(画面表示)すると、カーソルの位置指定や1行挿入、1行削除などの機能を使うことができます。

この仕組みのことをエスケープシーケンスというのです。

なかでも、カーソル位置指定のエスケープシーケンスをもっともよく見かけます。たとえば、座標(x,y)に"*"という文字を表示するのなら、

PRINT CHR$(27)+"Y"+CHR$(y+32)+CHR$(x+32)+"*"

※x、yには数値が入る

というPRINT文でカーソル位置指定から表示までができます。大文字のYのあとは、じっさいには直接そのキャラクタコードを持つ文字をつなげることが多いので、もっと短くなります。ほかのエスケープシーケンスも使ったサンプルプログラムを下に掲載しているので、打ちこんで試してみてください(リスト2)。

⮉リスト2の実行中画面。このパターンがそのまま上下にくりかえし動く

◉リスト2 エスケープシーケンスのサンプル

```
100 SCREEN Ø:WIDTH 4Ø:KEYOFF
110 E$=CHR$(27) 'ｴｽｹｰﾌﾟ ｺｰﾄﾞ
120 'ﾔｼﾞﾙｼ ｦ ｶｸ
130 PRINT E$+"YØ3*";
140 PRINT E$+"Y12***";
150 PRINT E$+"Y21*****";
160 PRINT E$+"Y33*";
170 PRINT E$+"Y43*";
180 'ﾔｼﾞﾙｼ ｦ ｳｺﾞｶｽ
190 LOCATE Ø,Ø
200 FOR I=Ø TO 3
210  FOR J=Ø TO 3:PRINT E$+"M";:NEXT
220  FOR J=Ø TO 3:PRINT E$+"L";:NEXT
230 NEXT
240 FOR I=Ø TO 12:PRINT E$+"M";:NEXT
250 FOR I=Ø TO 12:PRINT E$+"L";:NEXT
260 GOTO 18Ø
```

【解説】●行100=画面初期設定●行110=E$にエスケープコードを入れる●行120〜170=座標指定のエスケープシーケンス(Y××の部分)を使って矢印パターンを表示●行190=カーソルを(0,0)に移動●行200〜230=4行上へスクロール(ESC+"M"は1行削除の機能)と4行下へスクロール(ESC+"L"は1行挿入の機能)を4回くりかえす●行240〜250=同様に、13行の上スクロールと下スクロール●行260=行180へもどる

Q8 PEEK、VPEEK、POKE、VPOKEってなんですか。

A8 RAMやVRAMのあるアドレス相手に、データを読み出したり、書きこんだりする関数や命令です。

基本的にPEEKには「のぞく」という意味があり、POKEには「つつく」という意味があります。

BASICの命令語としては、
PEEKは、
PEEK(〈アドレス〉)
で、メモリの指定したアドレスにあるデータを与えてくれる関数(0～255の値になる)。

POKEは、
POKE〈アドレス〉,〈データ〉
で、メモリの指定したアドレスに、指定したデータを書きこむ命令。

それぞれの頭に「V」が付いたものは、VRAMに対する関数と命令になります。

Q9 あと何かいいことがあったら教えてください。

A9 BASICでテキストファイルを表示させる方法を教えましょう。

マニュアルに書いてなくて、しかもいいことっていうのはけっこうあります。いろいろあるなかで、かんたんに試せてちょっと不思議で楽しいものに「CON」という特別なファイル名があります。

これを使うと、BASICモードでも、ディスクのなかに入っているテキストファイルを画面に表示することができるのです。

MSX2+やターボRの人なら、
CALL KANJI
をあらかじめ実行しておいて、今月号の付録ディスクをセットし、
COPY "BCLN-3.C" TO "CON"
を直接実行してください。すると、「B：」の元データ(いろいろな制御記号と文書の混じったもの)が画面にどんどん表示されていきます。CTRL+Sを押すたびに表示の停止・再開もできます。

⬆3月号のB：のファイル「BCLN－3.C」を画面に表示すると

漢字が使えない機種は上記の実験をすると、たくさんのトランプマークが表示されるでしょう。それは漢字のデータです。

テキストファイルのほか、アスキーセーブされたBASICファイルも同様にして表示できます。

Q10 あっそうだ。A=USR(Ø)とかいうユーザー関数って何ですか。

A10 それはマシン語プログラムを呼び出すためのものです。

Mファンでは「USR関数」と表記していますが、これはいわゆる関数ではありません。DEFUSR～で指定したアドレス以降にマシン語プログラムを書いておき、それを呼び出すためのものです。

形式上は関数なので、使うときはA＝USR(Ø)などの形で使います。たいてい、カッコのなかの数値(引数)や得られる値には意味のない場合が多く、「使う」だけで、マシン語プログラムを呼び出せるのです。

もちろん、マシン語プログラムには、BGMを演奏したり、なにかグラフィックをかいたり、高速に計算したりといろいろなものがあるわけで、ここから先はいろいろですが、多くの場合、マシン語からふたたびBASICにもどって、変数になにかてきとうな値を代入して、次のステートメントを実行するというパターンがほとんどです(図7とイラスト参照)。

ときには、引数の値や関数から得られる値に意味のあるケースもありますが、それはUSR関数に備わった機能ではなく、呼び出されたマシン語プログラムがかってにやっていることです。

また、USR関数は、マシン語プログラムを書きこんでいないのに使われることがありますが、それは、特定のBIOS(ROMに入っているマシン語サブルーチン集)を呼び出しているのです。

●図7 USR関数

…A=USR (Ø)：PRINT A
マシン語プログラム

A = USR (Ø): PRINT A～
入力 Ø →
USR
マシン語プログラムを呼び出す
etc.
出力 Ø ←
AにØが代入される
けっきょく、数値に関してはなんにもしてない(ことがよくある)
A = USR(Ø): PRINT A～
そして BASICのプログラムはなにごともなく続いていく

STEP MORE　CP命令を使ったサーチプログラム

ここではＣＰ命令の応用としてデータサーチについて説明する。

サーチの方法

求めるデータ列の最初の１バイト目をサーチ開始アドレスの先頭から捜していく。

データ列の１バイト目とおなじデータが見つかったら、２バイト目と３バイト目がデータ列の２バイト目、３バイト目とおなじかどうかチェックする。違っていたら、１バイト目が見つかったアドレス＋１からサーチをやり直す。

このような手順を３バイトすべてが一致するか、範囲内に求めるデータ列がないことを確認するまで繰り返す。

図１をもとに、具体例を示そう。

まず、サーチデータ１バイト目〝M″をＤ０００番地から捜していく。このときＤ０００番地に〝M″があるので、２バイト目〝S″とＤ００１番地の内容を比べる。ここでデータが一致するので、３バイト目〝X″とＤ００２番地の内容を比べる。しかしここでは一致しないので、〝M″をＤ００１番地から捜していく。

Ｄ００４番地に〝M″があるので、〝S″とＤ００５番地の内容を比べる。ここでは一致しないので、今度は〝M″をＤ００５番地から捜していく。このとき〝M″がＤ００５番地にあるので、〝S″とＤ００６番地の内容を比べる。ここで一致するので、〝X″とＤ００７番地の内容を比べる。ここでも一致し、３バイト全てが一致したので、サーチが終了する。

CPＩR命令

このサーチデータの１バイト目を捜すのに最適なマシン語命令がある。それがCPＩR命令だ。

この命令については表２と図２を参考にしてほしい。

◉図1　サーチ領域の例

アドレス	D000	+1	+2	+3	+4	+5	+6	+7	+8	+9
メモリの内容	M	S	K	I	M	M	S	X	T	O

サーチ開始アドレス：DØØØ番地
サーチ領域の長さ　：10バイト
サーチするデータ　：'MSX'

サンプルプログラム

１バイト目をCPＩR命令で捜し、２、３バイト目はCP命令で確認するという、わかりやすさに重点をおいたプログラムだ。

いつものようにSimple ASMでアセンブルしたら、

>GD000,D010

で走らせてみよう。データが見つかれば〝F″と表示される。

また実行後にレジスタの内容を表示してくれるので、見つかったときのＩＹレジスタの値がそのアドレスとなる。確認は、例えばアドレスが１４１４Ｈなら、

>D1414

でメモリダンプしておこなう。

このプログラムでは３バイトサーチしかできないが、行1170〜1190および行1200〜1220がループ向きの構造なので、何バイトサーチにも改造できるだろう。また、CPＩR命令、またはCP命令のみでプログラムを組むことも可能だ。

◉リスト2　サーチのサンプルプログラム　NMH-4-2.SIM

```
1000          ORG      0D000H
1010 CHPUT:   EQU      00A2H
1020 ASM:     EQU      9000H
1030 STARTA:  EQU      0000H           ：サーチ開始アドレス
1040 LENGTH:  EQU      8000H           ：サーチ領域の長さ
1050          LD       IX,SDATA        ：サーチデータ格納アドレス
1060          LD       HL,STARTA
1070          LD       BC,LENGTH - 2
1080          CALL     SEARCH          ：サーチルーチン呼び出し
1090          CALL     Z,FOUND         ：もし見つかればFOUNDへ
1100          JP       ASM             ：アセンブラのシステムへ
1110 ;
1120 SEARCH:  LD       A,(IX + 00H)    ：1バイト目をAレジスタへ
1130          CPIR
1140          RET      NZ              ：指定領域内になかったらRET
1150          PUSH     HL
1160          POP      IY              ：IY←HL
1170          LD       A,(HL)          ：2バイト目をAレジスタへ
1180          CP       (IX + 01H)      ：2バイト目のチェック
1190          JR       NZ,SEARCH       ：違っていればSEARCHへ
1200          LD       A,(IY + 01H)    ：3バイト目をAレジスタへ
1210          CP       (IX + 02H)      ：3バイト目のチェック
1220          JR       NZ,SEARCH       ：違っていればSEARCHへ
1230          RET
1240 ;
1250 FOUND:   LD       A,'F'
1260          CALL     CHPUT           ：'F'表示
1270          DEC      IY              ：見つかったデータの先頭アドレス
1280          RET
1290 ;
1300 SDATA:   DEFB     0CDH,0A7H,0FFH  ：3バイトサーチデータ
1310          END
```

◉表2　CPIR命令

CPIR

セットアップ：
ＨＬレジスタ←サーチ開始アドレス
ＢＣレジスタ←サーチ領域の長さ
Ａレジスタ　←サーチするデータ

動作：
Ａレジスタと（ＨＬ）で示すメモリの内容を比較し、
ＨＬ←ＨＬ＋１、ＢＣ←ＢＣ－１をおこなう
以上の動作を、Ａ＝（ＨＬ）またはＢＣ＝０が成り立つまで繰り返す

フラグの変化：
Ａ＝（ＨＬ）で終了した場合、Ｚフラグがセットされる
ＢＣ＝０で終了した場合、Ｐ／Ｖフラグがセットされる
※Ｃｙフラグは変化しないので注意

注意：
ＨＬ←ＨＬ＋１とＢＣ←ＢＣ－１の計算は、Ａ＝（ＨＬ）が成り立つかどうかに関係なくおこなわれるので、Ａ＝（ＨＬ）が成り立ったときのＨＬレジスタの値（発見アドレス）を得たいときは、ＣＰＩＲの実行後、ＤＥＣ　ＨＬをおこなって算出する

◉図2　リスト2でのCPIR命令

BASICテクニック

多色刷りへの7つのステップ

多色刷りとはなにか。SCREEN1をカラフルに使うテクニックのことだ。それはむずかしいのか。いや、やさしい。それを使うのは面倒ではないのか。……んん、データが多いからちょっと面倒かな。

多色刷りは、SCREEN1の色の能力を192倍に拡張する魅力的なテクニックだ。あまりにもその効果が大きく、しかも、しばしばマシン語プログラムと同時に使われるため、難解なマシン語がからんでいるように思われがちだが、じつはマシン語の知識はまったくいらない。

そのかわり、赤青鉛筆で絵をかくよりも、「しまじろうの虹色お絵描きセット」でかくほうが手間がかかるのとおなじように、多色刷りはデータの用意がちょっと面倒だ。しかし、きれいなキャラクタが作れる。

理論的背景はちょっとむずかしいが方法はかんたん。7つのステップでカラフルパワフルな多色刷り技法に接近してみよう。

STEP 1 SCREEN1は32バイトしかカラー情報を持てない

SCREEN 1の文字の色は、ドット(前景色)とドットのない部分の色(背景色)に分かれる。ということは、最低でも2色ぶんの情報が必要だ。

1色の情報は、カラーコードで0～15、つまり4ビット。したがって、2色なら8ビット＝1バイト。文字は256種あるので、文字全体の色情報は256バイト……のような気がするが、図1を見てほしい。SCREEN 1の全文字の色情報は、中ほどにあるカラーテーブル(32バイト)にしかない。

SCREEN 1では、8文字ずつが色情報を共有しているのだ。

リスト1でそれを確かめよう。まず、全文字が画面の上部に表示され、メッセージが出る。そこでスペースキーを押すと、カラーテーブルの1バイトだけを書き換える。その結果は……8文字の色がいっぺんに変わってしまうのだ。

↑1バイト変更しただけで8文字の色が変わる

●リスト1　カラーテーブルを1バイト書き換えてみると

```
10 SCREEN 1:COLOR 15,4,7:WIDTH 32
20 FOR I=0 TO 255
30   VPOKE I+&H1800,I
40 NEXT
45 LOCATE 2,9:PRINT "hit space key"
50 IF STRIG(0)=0 THEN 50
60 VPOKE ASC("A")¥8+&H2000,&HF8
```

●図1　SCREEN1のVRAM構成

STEP2 SCREEN1にSCREEN2の皮をかぶせるBIOS

SCREEN 1は、前項のように8文字が1バイトの色情報を共有するという、おおらかな色付けをしているが、そんなSCREEN 1の性格をガラッと変えるためにいかにもとってつけたようなBIOSがある。

それがアドレス&H 7 Eだ。

DEFUSR=&H7E：A=USR(0)

を実行すると、BASIC上はSCREEN 1でも、VRAMでのデータの処理パターンはSCREEN 2になってしまう。

いきなり、これを試すと、文字がぜんぜん見えなくなってしまったりしてなんだかわけがわからなくなるが、それは必要なデータを送っていないからなのだ。

では、必要なデータとは？

STEP3 SCREEN2は6144バイトのカラー情報を持つ

図2はSCREEN 2のVRAM構成だ。図1と比べてみよう。SCREEN 1とSCREEN 2とは、BASICから見るとずいぶんちがう感じがするが、VRAMの構造を比較すると、よく似ている。ただ、大きさがちがう。

パターンジェネレータテーブルは3倍、カラーテーブルはあまりにも差があるので何倍かもわからないくらい大きい(192倍だが)。

SCREEN 2は、SCREEN 1では使われていなかった部分もしっかり使って、表現力を高めている。だから、SCREEN 2では、四角をかいたり、円をかいたり、好きなところにドットをかいたりできるのだ。

ただ、SCREEN 2は、SCREEN 1のようには文字を表示できない。あっちを立てればこっちが立たずというやつである。

◉図2　SCREEN2のVRAM構成

パターンジェネレータテーブル
0～&H17FF(6144バイト)

画面を上中下3つのエリアに分け、各エリアをさらに256の区画(各8×8ドット)に分けてグラフィックの状態(ドットの有無)を記憶している。横8ドット1ラインのドットの状態を2進数8桁(1バイト)で表し、8バイトで1区画のデータとなる。さらに、この区画はエリアごとに0～255のコードがつけられていて(パターン名称テーブル参照)、テキスト文字のキャラクタコードとパターンデータの場合とほぼ同様の関係を持つ。

パターン名称テーブル
&H1800～&H1AFF(768バイト)

3つのエリアに分かれたパターンジェネレータテーブルに対応して、ここも3つのエリアに分かれる。ただし、通常状態では、どのエリアも順に0～255のデータが固定されている(だから、パターンジェネレーータテーブルの状態がそのまま画面に対応している)。ここのデータを変更すると、そのエリア内にかぎり区画単位でグラフィックが変化する。

パレットテーブル
&H2020～&H203F(32バイト)

スプライトアトリビュートテーブル
&H1B00～&H1B7F(128バイト)

カラーテーブル
&H2000～&H37FF(6144バイト)

カラーテーブルの1バイトは、パターンジェネレータテーブルの1バイトに対応して、その1ラインの、ドットの色(前景色)と地の色(背景色)を記憶している(1バイトの上位4ビットが前景色、下位4ビットが背景色を指定する)。ここのデータを変更すると、そのデータに対応する1ライン(8ドット)の前景色と背景色が変化する。

スプライトジェネレータテーブル
&H3800～&H3FFF(2048バイト)

0
エリア1
&H800
エリア2
&H1000
エリア3
&H1800
&H2000
エリア1
&H2800
エリア2
&H3000
エリア3
&H3800
&H3FFF

STEP4 多色刷りのVRAM構成はSCREEN2とうりふたつ

で、また、STEP 2の話にもどる。SCREEN 1で例のBIOSを呼ぶとVRAMの構成が72ページの図1から下の図3へと劇的に変化する。

この図3は、左の図2の構成とよく似ている。よく似ているどころか、構成的にはまったくおなじだ。しかも、BASIC上はSCREEN 1のまま。これが、今月のテーマである多色刷りモードだ。

SCREEN 1とほぼおなじようにキャラクタを扱えて、しかもSCREEN 2とおなじVRAM構成を持つ。大きなパターンジェネレータテーブルと大きなカラーテーブルは、多色刷りのキャラクタを作成・表示するのには欠かせない領域だ。

◉図3　多色刷りのVRAM構成

パターンジェネレータテーブル
0～&H17FF(6144バイト)

画面全体(32桁×24行)を8行ずつ、3つのエリアに分け、それぞれのエリアに表示される文字のパターンデータを持つ。
【公式】ある文字のキャラクタコードをa、表示されるエリアをnとすると、その文字のパターンデータのあるアドレスは、
a＊8＋(n−1)＊&H800から始まる8バイト

パターン名称テーブル
&H1800～&H1AFF(768バイト)

機能はSCREEN1とまったくおなじ。
【公式】座標(x, y)に対応するデータがあるアドレスは、
&H1800＋x＋y＊32

パレットテーブル
&H2020～&H203F(32バイト)

スプライトアトリビュートテーブル
&H1B00～&H1B7F(128バイト)

カラーテーブル
&H2000～&H37FF(6144バイト)

表示エリア別に1セットずつあり、それぞれで1文字につき8バイトのカラー情報を記憶する。1バイトのデータは、1ライン(8ドット)の前景色×16＋背景色で表される。
【公式】ある文字のキャラクタコードをa、表示されるエリアをnとすると、その文字のカラーデータのあるアドレスは、
a＊8＋&H2000＋(n−1)＊&H800から始まる8バイト

スプライトジェネレータテーブル
&H3800～&H3FFF(2048バイト)

0
エリア1
&H800
エリア2
&H1000
エリア3
&H1800
&H2000
エリア1
&H2800
エリア2
&H3000
エリア3
&H3800
&H3FFF

STEP5 なぜパターンジェネレータテーブルは3倍必要なのか

ふつうのSCREEN 1で表示される文字は8×8ドット。パターンデータは1文字につき、8バイト必要だ。この8バイトに文字数(256)をかけたものがSCREEN 1のパターンジェネレータテーブルの大きさ(2048)である。

一方、多色刷りモードでの文字も、8×8ドットで、必要なパターンデータは1文字につき8バイト。文字もおなじく256種類しかないのに、なぜパターンジェネレータテーブルは3倍も必要なのか。

それは、多色刷りモードには3つのエリアがあるからだ。見た目はふつうの画面でも、上・中・下と3分割され、それぞれのエリアに表示される文字ごとにパターンデータが必要だ。つまり、3セット必要なのだ。

だから、おなじ文字(キャラクタコード)でも、パターンジェネレータテーブルの設定がちがっていれば、表示される位置によってキャラクタの形が変わる。

リスト2は、多色刷りのパターンジェネレータテーブルを実感するためのものだ。「@」のパターンが変化するのに注目。

リスト2を実行すると「A」と「@」が下に降りていくが……

●リスト2　エリア1から3を横断していく「A」と「@」

```
100 SCREEN 2:COLOR 15,4,15:CLS
110 SCREEN 0:WIDTH 80:WIDTH 40
120 SCREEN 1:DEFUSR=&H7E:A=USR(0)
130 FOR I=0 TO 7
140   VPOKE ASC("@")*8+&H800  +I,255
150   VPOKE ASC("@")*8+&H800*2+I,36
160 NEXT
170 LOCATE 10,1:PRINT "A"SPC(10)"@"
180 FOR I=0 TO 20
190 LOCATE 0,0:PRINT CHR$(27)+"L";
195 FOR W=0 TO 100:NEXT
200 NEXT
210 CLS:GOTO 170
```

●図4　おなじ文字でもエリアがちがえばパターンがちがう

STEP6 192倍になったカラーテーブルは文字をどう色づけるか

では、192倍のカラーテーブルはどうなっているのか。

まず、第1に3セットある文字のパターンデータ用にカラーデータも3セットある。

すると画面を3分割したエリアごとに2048バイトのカラー情報が割り当てられることになる。これでもまだ相当多い。

1文字あたりの数値を出すと、8バイト。1文字は8×8ドットだから、横1ライン(8ドット)に付き、1バイトのカラー情報が割り当てられている計算になる。

じっさい、多色刷りでは、文字のパターンの横1ラインごとに色情報を持っている。

図5は、大文字の「A」を多色刷りで、わざといろんな色で彩色してみた例だ。

エリア1の「A」をこの図5のような状態に色付けするプログラムがリスト3だ。

図5の前景色と背景色のデータをそのまま行180、190に置き、それを行150で読みこみ、VRAM用のカラーデータにしている。ここを注意してみれば、多色刷りのカラーテーブルの仕組みが具体的にわかってくるだろう。

ちなみに、このプログラムを実行すると、悪趣味な色付けをされたA以外の文字は見えなくなってしまう。SCREEN 0に復帰すればば文字はすぐに見えるようになるので、とりあえずREM文で入れてある行120を「'」なしで打ちこんでおけば、F1キーでSCREEN 0にもどれる。

リスト3によって無理やり多色刷りされたエリア1「A」の拡大写真

●リスト3　「A」をてきとうに多色刷りしてみる

```
100 SCREEN 1:COLOR ,,0
110 DEFUSR=&H7E:A=USR(0)
120 'KEY1,CHR$(21)+"SCREEN0"+CHR$(13)
130 LOCATE 10,5:PRINT "A"
140 FOR I=0 TO 7
150 READ F,B:C=F*16+B
160 VPOKE ASC("A")*8+I+&H2000,C
170 NEXT
180 DATA 4,11,5,10,6,9,7,8
190 DATA 8,7,9,6,10,5,11,4
```

●図5　多色刷りされた文字「A」のケース

前景色(ドットの色)	背景色(ドットのない点の色)										
4 (青)	11 (明黄)	→	11	11	4	11	11	11	11	11
5 (明青)	10 (暗黄)	→	10	5	10	5	10	10	10	10
6 (暗赤)	9 (明赤)	→	6	9	9	9	6	9	9	9
7 (水色)	8 (赤)	→	7	8	8	8	7	8	8	8
8 (赤)	7 (水色)	→	8	8	8	8	8	7	7	7
9 (明赤)	6 (暗赤)	→	9	6	6	6	9	6	6	6
10 (暗黄)	5 (明青)	→	10	5	5	5	10	5	5	5
11 (明黄)	4 (青)	→	4	4	4	4	4	4	4	4

STEP 7 4つの文字を使った多色刷りグラフィックの実例

RGBディスプレイならいいが、ビデオやRFで見ている場合、リスト3を実行したときのAなどほとんどなにがなんだかわからないはずだ。だから、現実的には色付けには限度があるし、必要に応じてキャラクタも大きくしたい。

といっても、1文字8×8ドットという制約だけはのがれようもないので、複数の文字を組み合わせて取り扱う、かんたんなテクニックもあわせて紹介しておこう。

リスト4の行140で定義されているA$は、文字列「ac」と「bd」をコントロールコードで接続して一時的なビッグキャラクタになっている。

行1000以降のデータが多色刷りの「面倒」を物語っている。たとえば「a」「c」の前景色はどのラインもおなじだが、1ラインごとにちゃんと前景色も含めたカラーデータを指定してやらなければならない。さらに全画面でこのキャラクタを表示するには、おなじことをもう2回、ほかのエリアのパターンジェネレータテーブルとカラーテーブルに書きこまなければならない。じつは、多色刷りでマシン語が使われることが多いのは、そのVRAM書きこみをマシン語でてっとりばやくすませているだけなのだ。

⇧リスト4を実行すると虹色の多色刷りくんが現れる。実際の大きさは4文字ぶん

●図6 図解「多色刷りくん」 ※数字(16進数)はカラーコード

'a' 'c' 'b' 'd'
1 2 3 4 5 6 7 8 9 A B C D E F
E 5

●リスト4 4つの文字を「多色刷りくん」にかきかえる

```
100 SCREEN 2:COLOR 15,0,0:CLS:SCREEN 1
110 'KEY1,CHR$(21)+"SCREEN0"+CHR$(13)
120 DEFINT A-Z
130 DIM P$(1),C$(1),P(1),C(1)
140 A$="ac"+STRING$(2,29)+CHR$(31)+"bd"
150 AD=ASC("a")*8
160 DEFUSR=&H7E:A=USR(0)
170 LOCATE 10,1:PRINT A$
180 FOR I=0 TO 15
190   READ P$(0),P$(1),C$(0),C$(1)
200   FOR J=0 TO 1
210     P(J)=VAL("&B"+P$(J))
220     C(J)=VAL("&H"+C$(J))
230     VPOKE AD+I+16*J,P(J)
240     VPOKE AD+I+16*J+&H2000,C(J)
250   NEXT
260 NEXT
1000 '*** pattern & color data
1010 DATA 11111111,11111111,E0,E0
1020 DATA 11111100,00111111,E1,E1
1030 DATA 11110000,00001111,E2,E2
1040 DATA 11100000,00000111,E3,E3
1050 DATA 11000000,00000011,E4,E4
1060 DATA 11001100,00110011,E5,E5
1070 DATA 10010010,01001001,E6,E6
1080 DATA 10000000,00000001,E7,E7
1090 DATA 10000000,00000001,E8,E8
1100 DATA 11000000,00000011,E9,E9
1110 DATA 11000010,01000011,EA,EA
1120 DATA 11100001,10000111,EB,EB
1130 DATA 11110000,00001111,EC,EC
1140 DATA 11111000,00011111,ED,ED
1150 DATA 01111000,00011110,5E,5E
1160 DATA 11111100,00111111,5E,5E
2000 '***       r g b ;=pallet
2010 COLOR=(1 ,7,0,0)
2020 COLOR=(2 ,7,3,0)
2030 COLOR=(3 ,7,4,0)
2040 COLOR=(4 ,7,5,0)
2050 COLOR=(5 ,7,6,0)
2060 COLOR=(6 ,7,7,0)
2070 COLOR=(7 ,5,7,2)
2080 COLOR=(8 ,3,6,3)
2090 COLOR=(9 ,0,6,4)
2100 COLOR=(10,2,6,7)
2110 COLOR=(11,1,5,7)
2120 COLOR=(12,0,4,7)
2130 COLOR=(13,1,3,6)
2140 COLOR=(14,3,0,6)
```

100 画面初期設定。おもに多色刷りモードに移行したときのためにあるていどの色設定をしておくため
110 リスト3の行120とおなじ
120 整数型宣言
130 P$(n)、P(n)はパターンデータの読みこみ用文字配列と書きこみ用数値配列、C$(n)、C(n)は同様にカラーテーブル用。nは0が左の文字、1が右の文字
140 A$に4文字の組み合わせ文字を定義する
150 ADは「a」のパターンデータ、カラーデータのオフセット(各テーブルの先頭番地との差)
160 多色刷りモードへ以降
170 多色刷りくんを表示
180～260 多色刷りくんのパターンおよびカラーデータ書きこみ
1000～1160 パターンデータおよびカラーデータ
2000～2140 各パレットコードをコード順で虹のようにグラデーションさせるためのパレット変更

BASICピクニック

→P108の「スーパー付録ディスクの使い方」参照

付録ディスク

そろそろ座標計算して動く線画アニメーションをやろうかと思っていたら、おなじテーマを持つ作品がファンダムに送られてきた。いただき。

線画のアニメーションツール

STEP-COUNT? 10

線画が変化していくステップ数を入力(写真上)したあと、まずスタートの線画Aをかき(写真左。青い線画)、いったん登録しておいてこんどは白い線で線画B(さらに左の写真)をかく。この線画A、Bをかくだけで、自動的にアニメーションをはじめる

この号向けのファンダム選考会に出品された線画アニメーションツール『MOVE−LINE』(作=うどん・てんどん/岡山・17歳)は、その名のとおり、線が動く作品だった。

マウスで一筆書きの線画Aをかき、それとは別におなじ線の数をもった一筆書きの線画Bをかくと、プログラムが座標を計算して、AからBへ連続的に変化していくアニメを見せてくれる。「マイケル・ジャクソンのビデオで人の顔が次々と変わっていくやつ」を見たことと、道を歩きながらふと「線が動いたらおもしろいかな」と思ったのがこの作品のきっかけだそうだ。

もっとも、このテーマそのものはむかしからある。市販ソフトで似たような狙いのものは以前からあるし、アマチュアプログラムの世界でも同種のものは多いにちがいない。ただ、Mファンへの投稿作品に関するかぎり、これが最初だと思う。

けっきょくファンダムでは採用基準に達しなかったので、特例として、BASICピクニックで取り上げることにした。じつは、いずれこのテーマを扱うつもりでいたのだ。

ファンダムではオリジナルを尊重するが、BASICピクニックはテクニックの紹介が主目的なので、操作性やプログラムの見栄えなど気になる点を大幅に改造して、右ページの『MOVE−LINE』(ファイル名MOVELINE.BP6で付録ディスクに収録)とした。変数名などもそうとう変更したのでずいぶん原作から変わってしまったが、根幹にあるアルゴリズムは原作のままだ。

ツールの使い方や原理は78ページで述べるが、とりあえず、この周囲にある写真を見てほしい。右上から左、左上から下へと経過していくのだが、「STEP−COUNT?」に対して数値を入力したあと、1枚目と2枚目の画面写真が線画A(青い描線)、線画B(白い描線)を設定しているところ。あとは自動的に、箱が開くようなアニメーションに移行する。

線画Aから線画Bまで、10ステップでアニメーションしていく(左上から右ページ右端まで)。このほか、いろいろな図形を試してほしい

◉リスト　2つの一筆書きから線画アニメを作り出す『MOVE-LINE』 (原案：うどん・てんどん　岡山・17歳)

```
MOVELINE.BP6

100 SCREEN 0:WIDTH 40:COLOR 15,0,0
110 DEFINT A-Z:DEFDBL D
120 KB=1           '== MOUSE:0 KEYBOARD:1
130 DEFFNA=STRIG(1) OR STRIG(0)
140 DEFFNB=STRIG(3)
     OR (PEEK(&HFBEB)AND4)=0  '== GRAPH
150 Z=100:DIM X(1,Z),Y(1,Z),DX(Z),DY(Z)
160 :
170 '=== START =======================
180 X=127:Y=105           '== CURSOL HOME
190 IF INKEY$<>"" THEN 190
200 INPUT "STEP-COUNT";ST
210 IF ST<1 THEN BEEP:GOTO 190
220 SCREEN 5,0:SPRITE$(0)="♠"
230 SETPAGE ,1:CLS:SCREEN ,0:SETPAGE ,0
240 :
250 '=== DRAWING =====================
260 ZZ=Z
270 FOR P=0 TO 1:C=0
280  GOSUB 520
    :IF FNA
      THEN X(P,0)=X:Y(P,0)=Y:BEEP
          :FOR W=0 TO 1:W=1+FNA:NEXT
      ELSE 280
290  GOSUB 520
    :FOR I=0 TO 1
     :LINE (X,Y)-(X(P,C),Y(P,C)),15,,XOR
    :NEXT
300  IF FNA
      THEN LINE -(X,Y),4+P*11:BEEP
          :C=C+1:X(P,C)=X:Y(P,C)=Y
          :FOR W=0 TO 1:W=1+FNA:NEXT
310  IF (NOT FNB OR P) AND C<ZZ THEN 290
320  BEEP:BEEP:BEEP:ZZ=C
330 NEXT
340 :
350 '=== ANIMATION ===================
360 FOR I=0 TO ZZ
     :DX(I)=(X(1,I)-X(0,I))/ST
     :DY(I)=(Y(1,I)-Y(0,I))/ST
    :NEXT
370 PUTSPRITE 0,(0,216):SETPAGE E,1-E
380 FOR I=0 TO ST
390  CLS
400  OX=X(0,0)+DX(0)*I
    :OY=Y(0,0)+DY(0)*I
410  PSET (OX,OY)
420  FOR J=0 TO ZZ
430   DX= X(0,J)+DX(J)*I
     :DY= Y(0,J)+DY(J)*I
440   LINE -(DX,DY)
450  NEXT
460  E=1-E:SETPAGE E,1-E
470 NEXT
480 IF FNA THEN 370
490 IF FNB THEN 170
500 GOTO 480
510 :
520 '=== DOT CURSOL OPERATION SUB =====
530 IF KB THEN 570 '=== SWITCH
540 P=PAD(12)        '== WITH MOUSE
550 XX=PAD(13):YY=PAD(14)
560 GOTO 590
570 S=STICK(0)       '== WITH KEYBOARD
580 XX=(S>1)*(S<5)+(S>5)
   :YY=(S=1)+(S=2)+(S=8)+(S>3)*(S<7)
590 X=(X+XX+256) MOD 256
   :Y=(Y+YY+212) MOD 212
600 PUTSPRITE 0,(X,Y-1),9,0
610 RETURN
```

■MOVELINE. BP6解説

100〜150　各種初期設定
100　画面初期化
110　整数型宣言、倍精度型宣言
120　使用入力機器設定
【KB：0以外のときキーボード使用、0のときマウス使用】
130〜140　ユーザー定義関数定義
【FNA：Aボタン、左ボタン、スペースキーの入力判定】
【FNB：Bボタン、右ボタン、GRAPHキーの入力判定】
150　座標管理用配列宣言
【Z：1枚の線画に使用可能な線数】
【X(n,m)、Y(n,m)：線画のm番目の頂点XY座標。nは0のとき線画A、1のとき線画B】
【DX(m)、DY(m)：線画Aのm番目の頂点が、それに対応する線画Bの頂点までアニメーションするための1ステップ単位の変化量】

170〜230　新しい線画用の初期化
180　カーソル位置の初期座標設定
190〜210　キーバッファクリア／変化のステップ数【ST】入力
220　スクリーン初期化／スプライトパターン設定(カーソル用)
230　VRAMのページ1クリア

250〜330　線画A、Bをかく
260　設定可能な点の数【ZZ】設定
270　Pのループ開始
【P：0＝線画A、1＝線画B】
【C：線画を構成する点のカウンタ】
280　ドットカーソル操作サブ呼び出し(行520、以下省略)／スペースキーが押されたらその点を配列X、Yの開始点として登録(Wのループによる連続入力防止ルーチン付き。以下も同様)、それまでは行280をくりかえす
290　点滅する線をかく
300　スペースキーが押されたら、その線を確定(線画Aでは青、Bでは白)／カウンタCを1増やす／配列X、Yにカーソル座標を登録
310　カウンタCが設定可能数ZZを超えてなければ行290へ(線画AのときはGRAPHキーもチェック)
320　(行310の条件からはずれたとき)3連発ビープ／ZZに現在のカウンタCの値を代入(線画A終了のときに線画Aの設定点数をBの設定可能点数とするため)
330　Pのループ閉じ

350〜500　アニメーション処理
360　配列DX、DY計算
370　スプライト消去／ページ切り換え(アクティブとビジュアル分離)
【E：ページ切り換え用、おもに働いているのは行460】
380　Iのループ開始(各ステップの線画をかく)
390　画面消去
400　そのステップでの開始点の座標【OX、OY】を設定
410　開始点にドットをセット
420〜450　線画をかく(つぎつぎに線をつなぎながらかいていく)
【DX、DY：線をかくための相対座標。直前のドットと現在のドットとの座標の差分】
460　ページ切り換え(表示)
470　Iのループ閉じ
480〜500　スペースキーが押されたら行370へ(おなじアニメのくりかえし)／GRAPHキーが押されたら行170へ(新しい線画へ)／(どちらかの入力があるまで)行480へ

520〜610　ドットカーソル操作サブ(キーボード・マウス兼用)

■MOVE－LINEの使い方

掲載・収録プログラムは最初キーボード(またはジョイスティック)用になっているが、行120をKB＝0にすればマウス用になる(ずっとマウス専用でかまわなければ、行120そのものを抹消してもいい)。

プログラムを実行してからの手順は、

①「STEP－COUNT?」ときいてくるので、線画A(最初にかく線画)から線画B(2番目にかく線画)まで何段階で変化させたいかを数値で入力。大きいほど細かくゆっくりとアニメーションする。10～20くらいが適当。

②ドットカーソルが現れる。スペースキー(Aボタン、左ボタン)で頂点を指定して線画Aをかいていく(描線は青)。直前に確定した点とドットカーソルとのあいだには仮の線(点滅)がひかれるので、それを参考にしてかく。GRAPHキー(Bボタン、右ボタン)を押すと線画Aが確定する。

※線を100本かくと自動的に終了・確定する。

③同様の要領で線画Bをかいていく(描線は白)。このとき、まえにかいた線画Aは青い線で見えているので、動きを考慮しながらかく。また、線画Aと線画Bの各線は設定されていった順で対応しているので線の対応関係も考慮する必要がある。線画Aとおなじ本数の線をかいた時点で自動的に終了・確定する。

④線画A、Bの手順がすむと、最初に設定したステップ数で線画Aから線画Bまでのアニメーションを表示する。

⑤もう一度おなじアニメーションを見たいときは、スペースキー(Aボタン、左ボタン)。はじめ(①)にもどりたいときは、GRAPHキー(Bボタン、右ボタン)。

※なお、線画のセーブ機能などはついていない。

■MOVE－LINEの原理

この線画アニメツールの動作原理は、その見た目に比べてじつに単純なものだ。

じつは、このプログラムは、線のアニメーションというより、点のアニメーションなのだ。

たとえば、点A(x0, y0)から点B(x1,y1)まで10ステップでアニメーションするにはどうしたらいいかを考えてみよう。

答えはかんたんだ。点A、Bを結ぶ線分ABを10等分する点を順に表示していけばいい。

点A、B間のX座標の差を10等分した値をdx、同様にY座標の差を10等分した値をdyとすると、アニメーションする点の座標は、X座標がdxずつ、Y座標がdyずつ増えていくことになる。

だから、i番目に表示される点の座標(X, Y)は、

X＝x0＋dx×i
Y＝y0＋dy×i

で求められるわけだ。

MOVE－LINEの線画アニメーションは、これとまったくおなじ方式で動いていく点をたんに線で結ぶことで、線画のアニメーションのように見せているのだ。

■線・面のアニメーション

実際の例にそって解説しよう。右上の図は、冒頭の写真例で使った立方体のかきはじめの部分で、P_0からP_7までを順に結んだものが線画Aの最初の部分、P_0からP_7'までを結んだものが線画Bの部分になる。

P_0からP_4まで(底面の部分)は重ねて設定しているので動かない。ここではP_5－P_5'、P_6－P_6'の線上を前述の原理で2つの点が動き、2つの点が動くたびにそれを含む8つの点を結んで線画をかいている(もちろん、まえの線は1回1回消しながら)。

すると、2つの点の動きは、3本の線分(P_4－P_5、P_5－P_6、P_6－P_7)の動きになり、それが立方体の1つの側面の動きに見えるというわけだ。

このように、点のアニメーションを、線・面のアニメーションに見せかけるのは、ワイヤフレーム(線画)のアニメーションに共通する原理だろう。

MOVE－LINEは、その原理をもっとも単純に純粋に実現してみせた作品なのだ。

■MOVE－LINEの改造点

1回にかく線画の複雑さが線100本まで、アニメーションはスタートとゴールの線画しか指定できない、一筆書きの線画しか扱えない、といった問題点は似たような処理をくりかえすていどの改造でクリアできるだろうが、現在のような配列だけの処理ではメモリの容量の問題につきあたるだろう。

その点は、互換RAMディスクやRAMディスク上にデータファイルを作って、それを配列がわりに使えばあるていどクリアできるだろう。また、その改造の途中で、データのセーブ／ロード機能を付け加えるのはかんたんだ。

例でいえば、点P_5から点P_5'にいたる軌跡は円になるはずだが、そのへんは線画Aから線画Bではなく、線画Aから線画B、線画C、線画D……と途中に経過する線画の数を増やしていけば、もっと細かい曲線的なアニメーションも理論的にはできるようになるはずだ。

また、スピードダウンをがまんすれば、色塗りされた面を含むアニメーションも、もうちょっとのくふうでできそうだ。

◉図　箱の展開アニメの一部

BASICテクニック

$F=ma$
りけい
$Ep=mgh$
$v=gt=9.8t$ $[m/s]$

MSX magazine 付録ディスク
→P108の「スーパー付録ディスクの使い方」参照

ディスプレイのなかでそれなりにリアルにはねまわるボールのシミュレーションを実現するために、どこでどう悩み、なにを捨て、なにを拾ったか。

家庭で悩む物理シミュレーション

#0　おわび

前号の『MOVE－LINE』のマウス対応部分にバグがありました。くわしくは、49ページを見てください。

#1　コンピュータの宿命

物理現象をシミュレートするのは、コンピュータの「宿命」のようなものだ。今回は、その宿命にとりつかれてみた。床と天井と左右の壁で囲まれた狭い部屋でボールをほうりなげ、そのあとのボールの動きをプログラムで表現してみようというのがテーマだ。

おなじテーマを持つプログラムがおそらくほかにあるだろうが、今回はあえてそうしたものをまったく参考にせずに、「BASICであるていどのプログラムを組める、物理学のしろうとが、家庭にこもったまま自力でシミュレーションのプログラムを作る」プロセスを苦しんでみることにした。

#2　モデル化のための捨象

まず、問題はどのていどのシミュレーションにするかという点だが、それよりも担当者はどのていどの物理法則を知っているかを考えたほうが話がはやい。

それが図1の、重力加速度、反発係数、摩擦係数である。それから有名な慣性。物体はほかから力が加わらないかぎり、一定の速度で運動するというやつだ。これを基本に、上記の3つの要素が加わってくる。

いろいろ考えている途中で、ボールの回転とか衝突時の歪(ゆが)みとかもけっこうボールの運動と関係ありそうだなとは思ったが、いずれにしてもそうした要素を扱う余裕はなさそうなので、ここではばっさり切り捨てることにした。こういうのを、かっこよくいえば、モデル化にあたっての捨象という。かっこわるくいうと、むずかしい問題はわざと知らないふりをする、か。

シミュレーションを作るというのは、大なり小なり、計算可能なモデル世界をディスプレイのなかに作るということなのだ。現実の物理法則をそのまま持ちこむことは化け物のようなコンピュータにだってできない。

#3　非ターボRの不幸

で、話は一気に飛んで、さんざん悩んで試行錯誤した結果が、72～73ページに掲載しているプログラムだ(付録ディスクに収録。操作方法は下記参照)。

実質的にターボRの高速モード専用のようになったが、MSX2／2＋でもテンポが乱れるが動くし、行160の「SC=100」の直後にある「'」を取れば、ゆっくり(ターボRの6分の1のスピード)ではあるが動きは均質になる。ただし、どちらも、床付近では、表示上のごまかしのため動きが乱れる(73ページの行660の解説参照)。

非ターボRのこの不幸は、このプログラムでまず最初にやった時間管理のせいだ。

このプログラムではシステム変数のTIMEが一定値に達するまで(ターボRは3、非ターボRは17)を1カウントとし、強制的な時間の区切りを作った。というのも、ボールの動きによって必要な計算量が変わるので、この区切りなしでやると、計算

『SIMBALL』の操作方法

MSX 2/2+ VRAM64K ※ただし、ターボRは高速モードで。また、ターボR以外の機種では、行160の1個目の「'」を取ってください(72ページの解説参照)。

①ボールの初期位置を決定
カーソルキーでボールを動かし、位置を決めたら、スペースキーを押す(ボールがグレーになる)。ESCキーでキャンセル。※ジョイスティック対応。スペースキーはAボタン、ESCキーはBボタン。

②投げる方向とパワーを決定
円の中心から伸びる線をカーソルキー左右で回転させ、投げる方向を決める(10度単位)。同時にカーソルキー上下でパワー調整(0～9)。スペースキーでボールが動く。ESCキーで①にもどる。

ボールは壁にあたるとやや速度を落として跳ね返り、床でバウンドしてやがて静止する(写真はプログラムを改造してボールの軌跡を残して撮影したもの)

◉図1 重力、反発係数、摩擦係数

◉重力　下方向への加速度
◉反発係数　跳ね返ったとき少しスピードが落ちる度合い
◉摩擦係数　全体にだんだんスピードが落ちてくる度合い

◉図2 床で跳ね返るボールの計算(行650〜670)

1カウント
T1
T2
※太線はボールの中心の軌跡
※図中の「XX＊T1」については、73ページ行660の解説参照
他の変数については下の本文で解説
現在のボールの位置　中心座標(X, Y)
XX＊T1
Y1
B1
G1
YY
G
R
G2
Y2
B2
1カウント後の位置
中心座標(A, B)
計算上の床
V(2)
行660で表示するボール
表示上の床
床がない場合の軌跡

のむずかしいところでは立ち止まり、やさしいところはすばやく通過するというような妙な動きになってしまうのだ。

もっとも、ターボRの場合はどんな計算もほとんど一瞬でやっているようなので、じつは時間管理をしなくてもたいして変わらないだろう。しかし、非ターボRでは場所によって3倍以上処理に時間がかかってしまう。

#4 角張った放物線

このプログラムでは、とにかく、床での跳ね返りが最大の焦点だった。プログラムでいうと行650〜670の部分だが、この問題さえなければ苦労は10分の1ですんだだろう。

床での跳ね返りが複雑なのはおもに重力のせいだ。それを解決するために1週間ほど悩んだすえにたどりついたのが、角張った放物線だ。

角張った放物線とは？

ここで扱うモデル世界に適用される物理法則で重力以外のものはすべて直線的な変化をするものばかりなので、単純なかけ算で処理できるのだが、重力による増分だけはいわゆる放物線的に変化していく。

今回のプログラムは、1カウントごとに、次の座標を計算して、点から点へワープしていく方式だが、その点はいちおう放物線上に置かれる。問題は、点と点のあいだで、床で跳ね返るなどの方向転換があったときだ。現実の世界では、その点から床、床から次の点までのあいだもやはり放物線的に変化しているはずなのだが、これをまともに相手にするのは、いろんな方面のつごうが悪い。

そこで、このモデルでは、

1カウント内では重力による増分も直線的に変化する………①

とした。つまり、点は放物線上にあっても、点と点は直線で結ばれていると考えるのだ。これが角張った放物線だ。そうすると計算がいっきに楽になり、いろんな方面のつごうがよくなる。

#5 B2を求めるプロセス

さて、1カウントぶんの座標増分をYYとすると、床さえなければ、

A=X+XX
B=Y+YY ……………………②

という計算で、次期座標(A, B)は単純に計算できる。

ボールが床についたときのY座標の限界をV(2)とすると、上記の式②の計算で仮の次期座標(A, B)を計算して、BがV(2)を超えていたら、どこか途中の地点Rで床にあたっていることになる。

ところで、YYはあらかじめ

YY=YY+G

という式で重力のぶんを加えられているので、式②にはすでに「ずっと下へ向かう」ことを前提にした重力による増分が入っていることになる。

そこで、重力による増分をRまでのG1とR以降のG2に分割し、さらに、Rまでの時間をT1、R以降の時間をT2、Rまでの重力以外の増分をY1、R以降の重力以外の増分をY2とする。さて、Rまでの全体の増分をB1、R以降の全体の増分をB2とする。それらの関係をまとめたものが図2だ。

求めるBは、図を見れば、

B=Y+B1-B2　または V(2)-B2

であることがわかるので、けっきょくB2の値がわかればいい。

以下、B2の値を求める計算の手順を追っていこう。

まず、B1の値はすぐに計算できる。

B1=V(2)-Y

このモデルでの特殊法則①から、B1に含まれるG1の比率は、YYに含まれるGの比率に等しい。Y1を中心に考えると、

Y1:YY-G=B1:YY
∴Y1=B1×(YY-G)/YY

YYもGも値はわかっているので、これでY1が求められる。

また1カウントのあいだにYY増え、T1のあいだにB1増えることからT1を求めると

1:YY=T1:B1
∴T1=B1/YY

さて、床にぶつかるときの速度は、B1／T1で、跳ね返った直後はその速度が逆方向に跳ね返る。それが残りのT2(これは1−T1に等しい)時間続いた場合の増分がY2だから、

Y2=B1/T1×(1-T1)

Y2から重力による増分の残りG2を引いたものが、目指すB2だ。まず、G2を求める。

G2=G-G1、G1=B1-Y1 から
G2=G-(B1-Y1)
∴B2=Y2-(G-(B1-Y1))

こうして、ようやく最終的なBが計算できる。

B=Y+B1-B2

いま追ってきたプロセスに、摩擦係数と反発係数の要素を加味すれば(これはかんたん)、床で跳ね返ったあとの正しいY座標が求められるわけだ。

こうやって床での跳ね返りに関して立てたアルゴリズムは、天井での跳ね返りでもまったくおなじ構造で通用する。

#6 人知を超えるプログラム

ところで、たったこれだけでボールはみごとに床で跳ね返り、徐々にバウンドを小さくしながら静止するのだが、正直にいって、なんだか知らないがうまくいった、というのが現在の心境である。まあ、プログラムというのはそのようにして人知を超えるものだが、今回はテーマのむずかしさと強引にたてた特殊法則があるだけにちょっと自信がない。今回だけではないが、心ある読者のご意見、ご批判をお待ちしております。　(コ)

◉リスト SIMBALL

※ターボR高速モード以外では行160を一部修正

```
SIMBALL .BP8

100 SCREEN 0:WIDTH 40
110 SCREEN 5,2:COLOR 15,4,0
120 OPEN "GRP:" AS #1
130 DEFFNA=STRIG(1) OR STRIG(0)
140 DEFFNB=STRIG(3)
     OR (PEEK(&HFBEC)AND4)=0 '== esc
150 DEFFNS=(STICK(0)+STICK(1)) MOD 9
160 C=3:SC=100':C=17:SC=3 '== ｼﾞｶﾝ､ｽｹｰﾙ
170 G=9.8*(C/60)^2*SC      '== ｼﾞｭｳﾘｮｸ
180 PR=3 + INT(SC/33)     '== ﾊﾟﾜｰﾘﾂ
190 E=.8                  '== ﾊﾈｶｴﾘ
200 FR=.98                '== ﾏｻﾂ
210 X=120:Y=100           '== ｼｮｷｲﾁ
215 A1=0:A2=0:B1=0:B2=0:Y1=0:Y2=0:T1=0
220 DIM H(2),V(2)         '== ｹﾞﾝｶｲ
230     H(1)=17:H(2)=238  '= ｽｲﾍｲ
240     V(1)=17:V(2)=194  '= ｽｲﾁｮｸ
250 DIM X(36),Y(36):A=ATN(1)/4.5
260 FOR I=0 TO 9          '== sin､cos
270   X(I)    = SIN(A*I)
    :X(18-I)= X(I)
    :X(18+I)=-X(I)
    :X(36-I)=-X(I)
280   Y(I)    =-COS(A*I)
    :Y(18-I)=-Y(I)
    :Y(18+I)=-Y(I)
    :Y(36-I)= Y(I)
290 NEXT
300 FOR I=0 TO 15           '== ball
310  A$="3567788888776530"
    :A=VAL(MID$(A$,I+1,1))
320  B=2^A-1
    :A1$=A1$+CHR$(B)
    :A2$=A2$+CHR$(B*2^(9-A)AND255)
330 NEXT
340 SPRITE$(0)=A1$+A2$
350 '
360 '=== ｼｮｷｲﾁ ｾｯﾄ
370 CLS:LINE (H(1)-8,V(1)-8)
          -(H(2)+8,V(2)+8),15,B
380 S=FNS
390   XX=(S>1)*(S<5)+(S>5)
400   YY=(S=1)+(S=2)+(S=8)+(S>3)*(S<7)
410 HS=1-FNB*7:GOSUB 730
420 IF FX=0 THEN X=A
430 IF FY=0 THEN Y=B
440 PUTSPRITE 0,(X-7,Y-8),15
450 IF NOT FNA THEN 380
460 IF FNA THEN 460 ELSE HS=1
470 '
```

◉解説

初期設定

100〜110　画面初期化。行100はリストを見るときに掲載リストどおりに見えるようにするため
120　グラフィック画面への文字表示準備
130〜150　ユーザー定義関数の設定。FNAはAボタン(ポート1)とスペースキーの入力判定。FNBはBボタンかESCキーの入力判定。FNSはカーソルキーかジョイスティックの方向キー入力
160〜215　変数初期化
C　60分のC秒が1カウント
SC　モデルのスケールサイズ。1メートルがモデルのSCドットに相当。行160の100の直後にある「'」を取ってリターンキーを押して修正すれば、非ターボR用の数値が設定されるようになる
G　重力加速度の9.8m/s²をスケールサイズと1カウントの大きさにあわせて調節している
PR　ボールの初速度を決めるときのパワーの倍率
E　反発係数(跳ね返るときに速さが落ちる割合)
FR　摩擦係数(空気摩擦の感じ)
X、Y　ボールの初期位置
A1、A2、B1、B2、Y1、Y2、T1　前ページの図2参照。行640〜670で使用。ここで初期化しておかないと、最初に無視できないタイムラグが生じてボールの動きがひっかかってしまう
220〜240　H(1)、H(2)は水平方向の限界座標、V(1)、V(2)は垂直方向の限界座標
250〜290　初速度設定のときに使用する三角関数値の計算と設定。10度きざみに、X(0)〜X(36)にはSIN関数値、Y(0)〜Y(36)にはCOS関数値が入る
300〜340　ボールのスプライトパターン定義。ボールは16ドット×16ドットのパターン。ボールを縦に半分に割り、1ラインごとにドットが右からいくつぶん埋まっているかをA$にデータとして設定している。そのドットの並び方に対応する数値(B)に変換し、A1$にはすなおに左半分のパターンを定義していく。同時に、そのパターンを左右反転させ、左へ1ドットシフトしたものを計算して、それを右半分のパターンとしてA2$に定義している

初期位置セット

370　画面消去。壁、天井、床をかく
380　カーソルキー(またはジョイスティック、以下省略)入力。Sに方向が入る
390〜400　増分計算。XXはX方向、YYはY方向の増分でSの値に応じて−1〜1の値をとる
410　HSは高速移動用変数。ESCキー(またはBボタン、以下省略)を押しているあいだはカーソルの動きが8倍速い。行730の限界チェックサブ呼び出し
420〜430　FX、FYは限界チェックサブで使うチェック用変数。現在のキー入力によって下記のように座標の限界を超えると、下記のような値をとる
X座標が左の壁から出た→FXが1
X座標が右の壁から出た→FXが2
Y座標が天井から出た　→FYが1
Y座標が床から出た　　→FYが2
440　ボールのスプライト表示
450　スペースキー(またはAボタン、以下省略)を押すまで行380から繰り返し
460　スペースキーを押したままでは次のプロセスへ行かないようにするための保護ルーチン。スペースキーが離されたらHSを1に初期化して次の処理へ

```
480 '=== ショソクト゛ セット
490 CIRCLE (X,Y),7,14:PAINT (X,Y),14
500 PUTSPRITE 0,(0,217)
510 S=FNS:IF S THEN BEEP
520 P=(P+(S=5)-(S=1)+10) MOD 10
530 D=(D+(S=7)-(S=3)+36) MOD 36
540 PSET (X-2,Y-3),0,TPSET
   :COLOR 6,14
   :PRINT #1,MID$("0123456789",P+1,1)
   :COLOR 15,4
550 LINE (X,Y)
      -STEP(10*X(D),10*Y(D)),10,,XOR
560 IF FNB THEN 370 '== キャンセル
570 IF NOT FNA
      THEN IF FNS
             THEN LINE -(X,Y),10,,XOR
                 :GOTO 510
             ELSE 560
580 IF FNA THEN 580
590 '
600 '=== ホ゛ール カ゛ ウコ゛ク
610 P=P*PR
   :XX=P*X(D):YY=P*Y(D) '== first set
620 TIME=0
   :XX=XX*FR:YY=YY*FR+G
630 GOSUB 730
640 IF FX THEN A1=H(FX)-X:A2=A-H(FX)
           :A =X+A1-A2*E
           :XX=-(A1+A2*E)
650 IF FY THEN B1=V(FY)-Y
           :Y1=B1*(YY-G)/YY
           :T1=B1/YY
        ELSE 680
660 PUTSPRITE0,(X+XX*T1-7,V(FY)-8),,FX
670            Y2=B1/T1*(1-T1)*E
           :B2=Y2-(G-(B1-Y1))
           :B=Y+B1-B2
           :YY=-B2/(1-T1)
680 X=A:Y=B
690 PUTSPRITE 0,(X-7,Y-8),15,0
700 IF FNB THEN 360
710 IF TIME<C THEN 710 ELSE 620
720 '
730 '=== ケ゛ンカイ チェック サフ゛
740 A=X+XX*HS:FX=-(A<H(1))-(A>H(2))*2
750 B=Y+YY*HS:FY=-(B<V(1))-(B>V(2))*2
760 RETURN
```

初速度セット

490　初速度設定用の灰色の円をグラフィックでかく
500　スプライト消去
510　カーソルキー入力(入力のたびにピープ音)
520　Pはパワーのレベル(0～9。9を超えると0にもどる)。カーソルキーの上下で増減する
530　Dは発射角度(0～35。0が真上、10度ずつ回転。35を超えると0にもどる)。カーソルキーの左右で左回り、右回りに回転する
540　パワーのレベル表示。灰色の円の中央に、暗い赤で数字を表示する
550　発射角度を表す棒を表示。円の半径より少し長い棒を角度に応じて表示する
560　キャンセル受け付け。ESCキーが押されたら行370(初期位置セットの始まり)にもどる
570　スペースキーの入力がなく、カーソルキーの入力があったら、行550でかいた棒を消し(XORで棒だけを消す)、行510(カーソルキー入力)へもどる。スペースキーの入力もカーソルキーの入力もなければ、行560(キャンセル受け付け)にもどる。スペースキーの入力があったら、次の行へ
580　スペースキーを押したままでは次のプロセスへ行かないようにするための保護ルーチン

ボールが動く

610　パワーレベルにパワー倍率をかけて、最終的なパワーを計算。発射角度を加味して、X、Y方向への初速度をセット
620　タイマー関数をゼロクリア(カウント開始)。X方向増分XX、Y方向増分YYを計算
630　行730(限界チェックサブ)を呼ぶ
640　左右の壁で跳ね返る処理。壁までの増分がA1、壁の向こうにはみ出たぶんがA2。A2は反発係数をかけて、かつ符号を逆にすれば跳ね返ったあとの次期X座標Aが求まる。次の回の計算(行620)のためにXXも計算しなおしておく(壁の向こうにはみ出したぶんに反発係数をかけ、符号を反転)
650、670　天井、床で跳ね返る処理。くわしくは71ページの本文と図を参照。跳ね返りのぶんだけに反発係数Eをかけている(行670の1行目Y2の計算)。
660　床で浮いたように見えるのを防ぐために、床についた瞬間のボールだけをカウントとカウントのあいだに特別に表示してごまかしている。図2のXX＊T1は、このときに使用するX座標を求める式。さて、スプライトパターン番号のパラメータのところに変数FXを置いているのはなぜでしょう？
680　新座標決定
690　ボール表示
700　キャンセル受け付け
710　カウント待ち

限界チェックサブ

740～760　仮に次期座標を計算し、その値が限界を超えていたらFX、FYにそれぞれ特定の値を設定して(そうでなければ0)もとにもどる

BASICピクニック

パソコン戦国時代の現在でも、やっぱりMSXは唯一の「ホームコンピュータ」だと思う。もっとMSXがホームコンピュータであるためのプログラム2題。

付録ディスク
→P108の「スーパー付録ディスク」参照

家計を考えるプログラム2題

MSXは計算機として、電卓にはない、きわめてすぐれた特徴を持っている。

たとえば、ついさっき、下の写真の縮小率を計算しようとして、ぼく(コ)はMSXに

?38.5/155

と打ちこんでリターンキーを押した(写真の横幅が155ミリ、本文1段ぶんの横幅が38.5ミリ)。結果は、

0.248387……

これではちょっと縮小しすぎなので、写真を2段ぶんの大きさに変更した。そこで上記の「38.5」の部分を83に変えてリターンキーを押す。

0.535483……

ここで、ぼくはレイアウトのために写真の53パーセント縮小コピーをとったりするわけだ。

このように、計算式の変更や計算のやりなおしが非常にやりやすい。今回は、家庭のなかで異能の計算機として働くMSXをより広く活用するためのプログラムを考えてみたい。

†

そこで、個人的に以前から疑問に思っていた、お金を借りたときの「元利均等方式」という返済システムを家庭のものにしてみようと思う。読者のなかにはお金を借りたことのない人も多いだろうが、このテーマは一種の数学の問題としてもけっこうおもしろい。それに、いつかはこの元利均等方式というヌエライクな(ヌエのような)仕組みのお世話になるにちがいないし。

†

お金を借りたら利子を付けて返す。これが資本主義社会の鉄則だ。全額をまとめて1回で返済する場合は話がかんたんだが、毎月少しずつ分割で返すとなると話が2つにわかれる。1つが、元金均等方式、もう1つが元利均等方式だ。

元金均等方式とは、元金(もともと借りたお金)を返済回数で均等に分割して返していく方式。じっさいには、毎月、元金返済分(一定額)に利子を加えて払うことになるが、利子は、まだ返してない元金(残債という)に利率(ひと月なら年率の12分の1)をかけてかんたんに計算できる。つまり、この方式はしろうとでも、毎月、いくらずつ返していけばいいのか計算できるわけだ(当然、毎月の支払額は減っていく)。

しかし、元利均等方式は一筋縄ではいかない。これは、元金返済分も利子分もまとめて、毎回一定額を返済していく方式だが、返済額の全体が一定であるかわりに、元金返済分は一定ではなく、そもそも、毎月の返済額がどのような計算式ではじきだされているのか、まったくわからない。今回の記事のために、さまざまな銀行ローンやノンバンクローンのパンフを調べたが、ついにこの計算について触れているものには出会わなかった。

それも当然だ。この計算は、きわめてむずかしい。

プログラムを動かすまえにちょっと次の問題を考えてみよう。

問 10万円を年率24パーセント(月あたり2パーセント)で借りたとする。毎月一定額を支払う元利均等方式で10回払いで返済するとすれば、毎月、いくら払えばいいでしょう。

本気で考えてほしい。

おそらくはほとんどの人がとちゅうでこんがらかってしまうだろう。そのこんがらかってしまう問題への答えが、69ページのプログラムだ。

【LOAN.BPXの使い方】

①画面に応じて、借入金(万円単位)、年率(パーセント単位)、回数(月払いで計算しているので

☞70ページにつづく

```
借入金(単位:万円)? 10
年率(%)? 24
回数(年×12)? 10

        11,700円
```

LOAN.BPXの画面。入力が終わるとまず適当な値を表示してからその数値を修正しはじめる

```
借入金(単位:万円)? 10
年率(%)? 24
回数(年×12)? 10

        11,133円

毎月返済額は 11,133円(端数切り上げ)
Ok
```

修正をくりかえしながら正解に到達する(ただし、1円未満は切り上げ)

その1　元利均等方式のローン返済計算

LOAN　　.BPX　　MSX2+以降

```
100 '***** ローン返済計算(元利均等)
110 '
120 '***  初期設定
130 '
140 SCREEN0:COLOR 15,4,7:WIDTH 39
   :CALL KANJI1:WIDTH 39
150 DEFSNG A-Z:DEFINT I-R
160 '
170 '*** データ入力
180 '
190 INPUT "借入金（単位：万円）";A
200 INPUT "年率(%)";E:E=E/1200  '月率
210 INPUT "回数（年×12）";N     '月数
220 DIM D(N):D(0)=0
 'D(K)=K回目の返済に含まれる元金返済分
230 DIM S(N):S(0)=0
'S(K)=K回目までのD(K)の和、S(N)=元金(A)
240 F=E*A+A/N        '毎月の返済額初期値
250 '
260 '***  D（N）とS（N）の計算
270 '
280 RS=-INT(LOG(F)/LOG(10))
290 FOR R=RS TO 4:D=10^-R
300   FOR I=1 TO N
310     D(I)=F-E*(A-S(I-1))
       :S(I)=S(I-1)+D(I)
320     LOCATE 9,6
       :PRINT USING"######,円";F*10000
330   NEXT
340   IF S(N)-A >0 THEN F=F-D:GOTO 300
350   F=F+D
360   LOCATE 9,6
     :PRINT USING"######,円";F*10000
370 NEXT
380 '
390 '*** 結果表示
400 '
410 PRINT
420 PRINT USING "毎月返済額は ######,円
(端数切り上げ)";F*10000
```

◉スピードアップのための改造 (ファイル名：SPEED.MRG)

```
290 FOR R=RS TO 1:D=10^-R
320     LOCATE 9,6
      :PRINT USING"###,千円";F*10
360   LOCATE 9,6
    :PRINT USING"###,千円";F*10
420 PRINT USING "毎月返済額は ###,千円(
端数切り上げ)";F*10
```

元利均等方式とは、毎回の返済額(元金返済分＋利子分)が一定で、そのなかに含まれる元金返済分の割合が変化していく方式。

■おもに画面モードの設定

140　SCREEN0の漢字モード1に設定。表示幅を39桁に設定しているが、漢字モードだと画面の左端が切れるようなので、その防止策

150　いったん数値変数全体を単精度型に宣言して、その一部を整数型に宣言しなおす。これはいくらかでも計算を速くするため(なにも宣言していないとすべて倍精度型とみなされる)

■必要なデータの入力と調整

190〜210　借入金、年率、回数の入力。年率は、パーセント単位で入力を受け付け、その後、実際の計算で使う月率の数値に変換している

A　借入金(元金)

E　月率

N　返済回数

220　配列Dの宣言。元利均等方式の焦点でもある、毎回変わる元金返済分を管理するための配列

230　配列Sの宣言。配列Dの途中までの合計値を管理

240　元金まるごとの1か月ぶんの利子(最高値)に、元金均等方式と同額の元金返済分を加えて、毎月の返済額のだいたいの見当をつける

F　仮の毎月の返済額(以下、「仮の返済額」)

■元金返済分の計算

280　仮の返済額Fの桁数を調べる

RS　仮の返済額Fの桁数をマイナスにしたもの。行290以降のループの初期値用

290　ループ(R)開始

D　仮の返済額Fの値を変化させる桁位置。計算のつごうで、マイナスは1万を超える桁。プラスは千以下の桁。

300〜310　仮の返済額Fによって全返済過程を計算し、元金の総返済額Fが正しい値かどうかを検証するループ(I)の開始

320　途中結果表示。表示のときは1万円単位に変換

330　ループ(I)閉じ

340　ループ(I)が終わって、返済しすぎていたらFを小さくする(Dで示される桁位置の数値だけを変更する)

350　(返済がたりないか、ぴったりだったら)小さくしたぶんをもとにもどす

360　調整した、仮の毎月の返済額表示

370　ループ(R)閉じ

■最終結果を表示する

410〜420　最終的に到達した毎月の返済額を表示(現実に通用しているものよりもおそらく1円高い)

金額や返済回数が多くなると、ターボRでもけっこう時間がかかる。そこで、結果は千円単位でわかれば実用的には十分。そういう人のために用意したプログラム(SPEED.MRGとして付録ディスクに収録)。アスキー形式でセーブされているので上のプログラムをロードしたあと
MERGE"SPEED.MRG"
を実行すればいい。

3か月なら3回、1年なら12回)を入力
②数字が減ったり増えたりするのをじっと見守る
③最後に結果を表示。ただし、1円未満は念のため切り上げてあるので、じっさいに払う金額より1円多いかもしれない。

†

もう1つ、電卓には絶対にまねできない計算機能として、表計算がある。

MSX用の表計算ソフトも市販されたが、あまりユーザーに(つまり、あなたたちに)相手にされなかった。値段的にも性能的にも魅力に乏しかったというのがおもな理由だろうが、1つには表計算ソフトというものが家庭でどんな役に立つのか、うまく伝えられなかったというのもあるだろう。

結論からいうと、表計算ソフトは家庭ではあまり役に立たない。というのも、家庭では表計算ソフトを使わなければならないような計算そのものがあまりないからだ。

しかし、いくつかの項目別予算を増減しながら、全体の予算のバランスをとる「やりくり」の作業には表計算ソフトの一部の機能が入りこむ余地がある。

やりくりでは、まえに書いた数字を消したり、新たに書きこんだり、合計を計算しなおしたりという作業が中心になるが、これこそ、コンピュータにさせたい仕事である。

そこで、複数の項目の予算を画面上で直接変更し、変更するたびに合計額を計算しなおすプログラムを作ってみた。

予算のバランスをとるためのものなので、バランサーという名前をつけたが、一般的にこの手のものをこう呼ぶわけではないので注意してほしい。

【BALANCER.BPXの使い方】
①目的に応じて、プログラムの冒頭にあるデータを変更する(データの意味などについてはリスト右側の解説参照)
②実行すると、各項目の金額と合計を表示する
③数値の変更は1桁単位

その2　やりくりの結果がすぐにわかるバランサー

BALANCER.BPX　MSX(RAM8K) MSX2/2+

```
10 '****** ﾊﾞﾗﾝｻｰ
20 '
30 '*** ｺｳﾓｸｽｳ(N),ﾃﾞｰﾀ ﾉ ｻｲﾀﾞｲｹﾀｽｳ
40 DATA 11         ,7
50 '*** ｼｮｼｷ(PRINT USING)
60 DATA "&          &   ¥########,"
70 DATA "&          & ¥##########,"
80 '*** ｺｳﾓｸﾒｲ      ,ｼｮｷﾁ
81 DATA ｼｮｸﾋ        ,50000
82 DATA ｼﾞｭｳｷｮ      ,150000
83 DATA ｽｲﾄﾞｳ･ｺｳﾈﾂ,15000
84 DATA ﾋﾌｸ        ,18000
85 DATA ﾎｹﾝ･ｴｲｾｲ   ,20000
86 DATA ｷｮｳｲｸ      ,50000
87 DATA ｷｮｳﾖｳ･ｺﾞﾗｸ,5000
88 DATA ｺｳｻｲ       ,5000
89 DATA ｺｳﾂｳ･ﾂｳｼﾝ ,15000
90 DATA ｻﾞｯﾋﾟ      ,10000
91 DATA ﾁｮﾁｸ       ,50000
100 '
110 '*** ｼｮｷｾｯﾃｲ
120 '
130 SCREEN 0:COLOR 15,4:WIDTH 40:KEYOFF
140 XL=14              '** ｶｰｿﾙ ﾋﾀﾞﾘｹﾞﾝｶｲ
150 X=XL:Y=0                '** ｶｰｿﾙ ｼｮｷﾁ
160 RESTORE 30:READ N,L:N=N-1
170 DIM K$(N),Y(N)          '** ｺｳﾓｸ,ｷﾝｶﾞｸ
180 RESTORE 50:READ F$,G$'** ｼｮｼｷ ﾖﾐｺﾐ
190 RESTORE 80
   :FOR I=0 TO N
     :READ K$(I),Y(I)
   :NEXT                 '** ｶｸｺｳﾓｸﾃﾞｰﾀ ﾖﾐｺﾐ
```

バランサーという名前はこの記事中でてきとうにつけた名前

■項目数、データの桁数、初期データ

40　予算のある全項目数と、各データの最大桁数(行160で読み出し)
60〜70　各予算表示用の書式と、合計値表示用の書式(行180で読み出し)。PRINT USING用
81〜　各項目の項目名(10字以内)と初期値

■画面設定、配列・変数設定

130　画面設定
140〜150　変数初期設定
XL　カーソル位置X座標の左側限界
X、Y　カーソルの座標(ここでは初期位置座標を設定)
160　項目数、最大桁数読みこみ
N　予算項目数(実際にはN−1して使用)
L　各データの最大桁数
170　項目用文字配列K$、項目別金額用の配列Y宣言
180　各項目データ表示用(F$)、合計額表示用(G$)の読みこみ
190　各項目データ読みこみ

BASIC テクニック

④変更したい数字にカーソルをあわせ、数値(1桁)を入力。同時に合計欄も自動的に変わる(数値がなく、空白になっている桁でも入力可能)

†

掲載プログラムは、すべて付録ディスクに入っているので、もちろん実際に使ってみてほしい。ただし、このプログラムが表示した結果によってなにか問題が起きたとしても、わしゃ知らんけんね。次は受験直前英単語帳ソフトだ。以下次号　(コ)

```
ショクヒ              ¥█  50,000
ジ゛ュウキョ          ¥  150,000
スイト゛ウ・コウネツ  ¥   15,000
ヒフク                ¥   18,000
ホケン・エイセイ      ¥   20,000
キョウイク            ¥   50,000
キョウヨウ・ゴ゛ラク  ¥    5,000
コウサイ              ¥    5,000
コウツウ・ツウシン    ¥   15,000
サ゛ッヒ゜            ¥   10,000
チョチク              ¥   50,000
=======================
ゴ゛ウケイ        ¥    388,000
```

BALANCER.BPXの画面。とりあえず入れてあったデータで架空の家庭の家計を表示。金額のカンマは自動出力

```
ショクヒ              ¥   40,000
ジ゛ュウキョ          ¥  150,000
スイト゛ウ・コウネツ  ¥   10,000
ヒフク                ¥    8,000
ホケン・エイセイ      ¥   10,000
キョウイク            ¥   30,000
キョウヨウ・ゴ゛ラク  ¥    5,000
コウサイ              ¥    5,000
コウツウ・ツウシン    ¥   10,000
サ゛ッヒ゜            ¥   10,000
チョチク              ¥   22,000
=======================
ゴ゛ウケイ        ¥    300,000
```

数値を変更すればすぐに結果が出る。写真は、いろいろ削ったあと「チョチク」の額を2万2千円まで抑えて合計を30万円に収めた瞬間

```
200 '** キーニュウリョク チェックヨウ モジ゛レツ CH$
210 CH$=CHR$(28)+CHR$(29)+CHR$(30)
      +CHR$(31)+"0123456789"
220 '
230 '*** テ゛ータ ヒョウジ゛
240 '
250 S=0:PRINT CHR$(11);
260 FOR I=0 TO N
270 PRINT USING F$;K$(I),Y(I)
280 S=S+Y(I)
290 NEXT
300 PRINT STRING$(23,"=")
310 PRINT USING G$;"ゴ゛ウケイ",S
320 '
330 '*** カーソル セイギ゛ョ
340 '
350 LOCATE X,Y:A$=INPUT$(1)
360 CH=INSTR(CH$,A$)
370 IF CH=0 THEN 350
380 IF CH=1 THEN IF X>XL+L THEN 350
390 IF CH=2 THEN IF X=XL   THEN 350
400 IF CH=3 THEN IF Y=0    THEN 350
410 IF CH=4 THEN IF Y=N    THEN 350
420 IF CH>4 AND VPEEK(X+Y*40)=ASC(",")
      THEN A$=","
430 PRINT A$;
440 X=POS(0)+(CH>4):Y=CSRLIN
450 IF CH=<4 OR A$="," THEN 350
460 '
470 '*** テ゛ータ ノ コウシン
480 '
490 K=L+1-(X-XL):K=K+(K>3)+(K>7)
500 Y$=STR$(Y(Y)):LY=LEN(Y$)
510 IF K>=LY THEN 550
520 MID$(Y$,LY-K,1)=A$
530 Y(Y)=VAL(Y$)
540 GOTO 230
550 Y(Y)=Y(Y)+VAL(A$)*10^K
560 GOTO 230
```

210　CH$の設定
CH$　キー入力チェック用文字列。最初から4文字ぶんはカーソル制御コード。5文字め以降は通常の数字になっている

■各データと合計の表示

250　合計値初期化。カーソルをホーム位置へ
S　合計値
260　表示用ループ(I)開始
270　書式に従って項目名とデータを表示
280　各データの合計
290　ループ(I)閉じ
300　23個の"="を表示。各種データと合計額との境を表示
310　合計値表示

■データ変更用カーソルの処理

350　カーソルをもとの位置にもどし、1文字のキー入力待ち(A$)
360　入力された文字の判別。入力された文字がチェック用文字列CH$のどの文字と一致するか(またはしないか)を調べ、一致する文字があればその文字のCH$での順番をCHに保存
CH　入力文字判別用。1～4のとき、カーソル制御コード、5以上のとき数字、0のとき、それ以外の文字を表す
370～410　カーソルの限界コントロール
420　カンマになっているところに数値を入力しても無効にする
430　A$表示(中身がカーソル制御コードだとカーソルが移動する)
440　カーソルのあった座標の保存。入力された文字が数字(CH>4)だと1字ぶん右に進んでしまうのでわざと1つもどす
450　入力された文字がカーソル制御コードか、あるいは、カンマの上に数字を入力したか(行420)であれば行350へもどる

■データの更新

490　桁位置の計算(数字についているカンマも考慮して計算)
K　更新するデータの桁位置
500　更新するデータの文字化。文字の長さを調べる
Y$　更新する対象となるデータの数字列
LY　Y$の長さ(数値の実際の桁数より1多い)
510　数字を入力された桁が更新する数値よりも大きければ行550へ
520　(上記以外)該当する桁の数字を入力された数字と置き換える
530　数字の数値化
540・560　行230へもどる
550　入力された数字の桁にみあった数を加える

BASICピクニック

俗に、英単語を3000語覚えていれば東大に入れ、6000語覚えていればニューヨークタイムスがスラスラ読めるという。英単語のたぐいを覚えるための簡易データベースの試み。

付録ディスク →P108の「スーパー付録ディスク」参照

家庭で作る英単語帳ツール

とつぜんだが、勉強のやり方には、大きく分けて2つのタイプがある。

1つは、参考書型。演繹(アプリオリ)型といってもいい。公式や概論や一覧表など、エッセンス的なものばかり勉強して、問題はほとんどやらない。勉強にかかる時間が少なくてすむが、実戦の経験がないのでテストでは苦しむことが多い。わたし(コ)はこのタイプだった。

2つめは、問題集型。これは帰納(アポステリオリ)型といっていい。とにかく具体的な問題を数多くやることでじわじわと本質に迫っていくタイプだ。わたしの知るかぎり、こういうタイプは思考力が骨太で社会的に成功する。

†

英単語を覚える場合、参考書タイプは単語帳で、問題集タイプは英文読解をしながら覚えようとするようだが、こと日本で英語のテストを受けるかぎりにおいては、英単語は単語帳で覚えるのがいちばんいいと思う。

文脈で覚えよう、などとばかなことがいえるのは、文脈のおそろしさを知らないのだ。文脈は人間の指紋のようにおなじものは1つとしてない。そんなもので単語を覚えようとしたらたいへんだ。頭のなかは少しずつ意味のちがう単語でいっぱいになり、爆発してしまうだろう。

†

BASICピクニック史上、前例にないほど長く、もったいぶった前振りだったが、今回は英単語帳ツールを作ってみた。

†

まず使い方から。

なるべくフォーマット直後のディスクに付録ディスクから
WORDNOTE.BPZ
WORD.DAT
の2つをコピーし、かならず書き込み可の状態にして使うこと。

WORDNOTE.BPZを実行すると、下の写真のようなメニュー画面が現れる。「新規」は新しい単語を追加すること、「めくる」はファイルの中の単語を見ること、「終了」はプログラム終了。

「めくる」で単語を見ながら、修正・抹消ができる。ただ、どのデータにも変更を加えないでいると変更・抹消のメニューが出ないので、抹消したい単語は必ずどこかを修正すること。

さらに「新規」では、「登録」(入力したものをファイルに書き込む)、「訂正」(入力したものに修正を加える)、「取消」の3つに分かれている。

「めくる」は、「ABC」(アルファベット順に見る)、「選択」(最初の2文字を指定して単語を検索する)、「乱数」(ランダムな順番で表示する)の3つに分かれる。

メニューの選択はカーソルの左右、決定はスペースキー。これが基本だが、単語をめくったときは1要素ごとにリターンキーを押していかなくてはならない。

†

じつは、このツールは前回のBASICピクニック用に作っていたものだったのだが、最後までバグが取れず、2か月先延ばしにしたのだった。

正直な話、今回ディスクに収録してある『英単語帳ツール』(WORDNOTE.BPZ)はたいしたものではない。が、ここにいたるまでは正味1か月かかった。

こんなものになんで1か月もかかったのか。その理由を説明することによって、こうしたデータベース的なものを作るときのカンどころを考察することができるだろう。

†

まず、シーケンシャルファイルかランダムファイルかという問題がある。

単語ごとに必要な文字量はちがうので、その点を考えれば、1レコードごとに長さを自由に変えられるシーケンシャルファイルがいいような気もする。

しかし、それ以上に、単語帳は、学習状況に応じて変化していくものだ。覚えてしまった単語は削っていき、新しい単語を追加していかなくてはならない。

```
英単語帳メニュー

□新規　■めくる　□終了
```

メニュー画面。まずおおまかな作業を選ぶ。単語追加は「新規」、単語の修正は「めくる」で単語をめくってからいつでもできる

```
【綴り】=====================<
beautiful
【品詞】=<
形容詞
【読み】(1行)
ビュー]ティフル
【意味】(2行)
美しい

SPACE=続行　ESC=中断
```

「めくる」を選ぶと、1つのレコードが終わるたびに、続行するかどうかをきいてくる。続行するならスペースキーを押す

なのに、シーケンシャルファイルは、追加することしかできないし(修正もできない)、レコードはおなじ順番でしか取り出すことができない。

そうなると、すべてのレコードがおなじ長さになってしまう(そのぶん容量をムダにくってしまう)が、自由に修正、追加ができ、読みだす順番にも制約のないランダムファイルのほうが適していることになる。

じっさい、単純なデータの記録以外は、すべてにおいてランダムファイルのほうがシーケンシャルファイルより使いやすい。

†

ランダムファイルのレコードは、FIELD文によって、レコードの中をいくつかに分割できる。単語帳の場合、①綴り(ツヅ)、②品詞、③読み、④意味(解説)の4つに分けることがまず考えられるが、じつはもう1つ必要な要素があった。

それは、1つ1つのレコードを管理するための整理番号的な要素だ。

たとえば、ある単語を削除した場合、その単語に使われていたレコードそのものは残っているので、どこかに「このレコードのデータは無効」だということを知らせるものが必要だ。

また、単語帳を作っていく場合、気がついた単語から入力していくので、じっさいにできあがる単語帳ファイルでは、単語が順不同で収められているだろう。しかし、せっかくコンピュータを使うのだから、ABC順に並べ替えたり、目的の単語を検索したりしたいものだ。そういうときのために、単語ごとに整理番号をつけられたら管理がしやすいだろう。

最終的に、この要素は「タグ」(荷札のたぐい)と名付けて、レコードの頭に置いたが、このタグの形にたどりつくまでに1か月のうちの25日を費やしたのであった。逆にいうと、このタグの作り方が悪いせいで、25日のあいださまざまなバグに悩まされたのだ。

最初の25日間に試みたタグの形式がどんなものであったかをここに書いても話が複雑になるばかりなので省くが、最終的に(70〜71ページに掲載している『英単語帳ツール』のプログラムで)どんな形になっているのかというと、なんと、単語の最初の2文字をキャラクタコードに変えた数値が入っているだけだ。また、単語を抹消したレコードのタグには&H7FFF(整数型の最大値)を入れている。

ついでに、英単語帳ツールのレコードの構造を解説すると、

①タグ　2バイト
②綴り　31バイト
③品詞　10バイト
④読み　39バイト
⑤意味　78バイト

となる。

②が31バイトなのは、「floccinaucinihilipilification」(29字、金銭蔑視)という単語が楽に入れられる大きさという気持ち。ほんとうは、「pneumonoultramicroscopicsilicovolcanokoniosis」(45字、肺塵症)まで入れられると完璧なのだが、そうなるとかなりのムダを作ることになるのでやめた。

④、⑤は、漢字モードにしたときの見やすさのためにWIDTHを39にしているため、39の倍数にしてある。

さて、①のタグの重要さを語るためには、このプログラムでファイル関係とは別に設けている2次元配列Rについて、語らなければならない。

†

ディスク上にできているファイルでは、かんたんにレコードを並べ替えることはできない。そこで、実際のレコード番号(ここではこれを実レコード番号と呼ぶ)とは別に、整理上のレコード番号(同様に仮想レコード番号と呼ぶ)を作ると、この仮想レコード番号に対応する実レコード番号を並べ替えることで、レコードを並べ替えたのとおなじ効果が得られる。

◎図「新規」部分のアルゴリズム

その仮想レコード番号と実レコード番号を対応づけるのが2次元配列Rだ。

具体的には、Vを仮想レコード番号とすると、1つのVに対して、配列Rは2要素あり、それぞれ、その単語の

R(V,0)……タグ
R(V,1)……実レコード番号

を表している。

管理は仮想レコード番号でおこない、じっさいにファイルからデータを読み出すときは実レコード番号を使って読み出す。

この場合、タグはその実レコード番号に入っている単語を代表するデータになっている。単語をアルファベット順(つまりタグ順)に並べ替えるときも、R(V,0)のデータを比較して、タグと実レコード番号をセットにして並べ替えておけばいいことになる(行1620)。

ところで、仮想レコード番号は1からはじまるので、R(0,0)とR(0,1)があまる。そこで、

R(0,0)……ソートする必要があるかどうか(1だと必要あり)
R(0,1)……使用レコード数(総レコード数のうち、削除した単語を除いたもの。有効な仮想レコード番号の最大値)

というふうに特別な意味を持たせた。上の流れ図は、参考までに新規単語を入力する部分だけを抜き出したものだが、行510、520をはじめ随所にこの2つが顔を出し、重要な役目を果たしている。

けっきょく、このプログラムは配列Rにつきるといっても過言ではない。データベース的なソフトでもっとも大切なのは、そのデータを効率的に管理できる配列を作りだすことだ。(コ)

英単語帳ツール ファイル名：WORDNOTE. BPZ（データファイル：WORD. DAT）

※ターボRの高速モードでは、行265のREM（'）を取れば作業時はRAMディスクで動作するようになります。

```
100 '<<<<<<<<< 英単語帳ツール >>>>>>>>>
110 '[[[[[[[[[[ 初期設定 ]]]]]]]]]]]
120 CLEAR 1000
130 SCREEN 0:COLOR 15,4,7:WIDTH 39
140 CALL KANJI1:WIDTH 39:KEYOFF
150 ON ERROR GOTO 2000
160 ON STOP GOSUB 1900:STOP ON
170 DEFINT A-Z
180 DEFFNESC=((PEEK(&HFBEC) AND 4)=0)
190 M=1000                    '単語最大数
200 DIM L(4),W$(4),T$(4),M$(4),R(M,1)
210 RESTORE 220
   :FOR I=1 TO 4:READ M$(I):NEXT
220 DATA
   【綴り】======================<,
   【品詞】=<,      【読み】（1行），
   【意味】（2行）
230 RESTORE 240
   :READ L(0),L(1),L(2),L(3),L(4),F$
   '      タグ，綴り，品詞，読み，意味，ファイル
240 DATA    2,  31,  10,  39,  78,WORD
250 D$="A:"      '"WORD.DAT"があるドライブ
260 F$=D$+F$+".DAT"
265 'RD=1:OPEN F$ AS #1:CLOSE
    :CALL RAMDISK(0):CALL RAMDISK(1000)
    :CALL CHDRV("H:")
    :COPY F$:MID$(F$,1,1)="H" 'RAMDISK
270 L=L(0)+L(1)+L(2)+L(3)+L(4)
280 '*** ファイル準備 ****************
290 C=0     '新規ファイル作成希望有無
300 CALL CLS:COLOR 8
310 PRINT "ファイル準備中":COLOR 15
320 OPEN F$ AS #1 LEN=L
   :FIELD #1,            L(0) AS W$(0),
      L(1) AS W$(1), L(2) AS W$(2),
      L(3) AS W$(3), L(4) AS W$(4)
330 N=LOF(1)/L
   :IF N THEN 410
        ELSE IF C THEN 440
340 '（単語のファイルがないとき）
350 CLOSE:KILL F$:CALL CLS
360 PRINT "このディスクにはデータがあり
ません。このディスクに新規ファイルを作
りますか。"
370 PRINT "□はい   □いいえ  □中止"
380 Y=2:X=0:GOSUB 1800:C=(X=0)
390 ON X+1 GOTO 300,400,1930
400 PRINT:PRINT "別のディスクを入れてス
ペースキーを押してください。"
   :FOR I=0 TO 1:I=-STRIG(0):NEXT
   :GOTO 280
410 '*** 配列Rの初期設定 ************
420 FOR I=1 TO N
     :GET #1,I:T$=W$(0)
     :R(I,0)=CVI(T$)
     :R(I,1)=I
   :NEXT
430 GOSUB 1600
440 '[[[[[[[[[[ メニュー ]]]]]]]]]]]
450 COLOR 1,14:CALL CLS
460 PRINT "英単語帳メニュー":PRINT
470 PRINT "□新規   □めくる  □終了"
480 Y=2:X=0:GOSUB 1800
490 ON X+1 GOTO 500,670,1900
500 '*** □新規 **********************
510 IF R(0,0) THEN GOSUB 1600
520 V=R(0,1)+1
   :IF V>M THEN GOSUB 2100:GOTO 440
        ELSE R(0,1)=V
530 ERASE T$:DIM T$(4)
540 GOSUB 1400
550 LOCATE ,11
   :PRINT"□登録   □訂正   □取消";
```

おもなプログラムと変数の解説

110 ■初期設定
150～160 割り込み先指定
180 ESCキー入力判定用関数の定義
190～200 M設定／各種配列宣言
M 処理可能な単語の最大数
L 各フィールドの長さ
W$ 各フィールド用バッファ変数
T$ バッファ変数に対応する処理用変数
M$ メッセージ用変数
R 仮想レコード管理用
265 ターボRでRAMディスクを使う場合の改造用プログラム（「'」を削除して、この行を生かすとフロッピーディスク内にあったファイルをRAMディスクにコピーし、以降、RAMディスクで読み書きする）
RD RAMディスク使用フラグ
270 レコード長Lの設定
280 ■ファイル準備
290 新規ファイル作成の希望の有無をチェックするための変数Cの初期化（0：希望有）
320 単語帳データのファイル準備
330 総レコード数計算／すでになにかレコードがあれば行410へ、なにもないが新規レコードを作っていれば行440へ
N 総レコード数（削除された単語も含む）
350 ファイルをクローズし、削除（ファイルがない場合も行320のOPEN文で強制的にファイルが作られているため）
380 メニュー選択サブ（行1800）を呼ぶ／「はい」を選べばCに－1を入れる
Y メニュー選択カーソルを表示するY座標
X メニュー選択カーソルを表示するX座標の10分の1
390 カーソル位置に応じて各行へ飛ぶ
400 メッセージ表示／スペースキー入力待ち／行280へ
410 ■配列Rの初期設定
420 ファイルのデータをすべて読み、Iを仮想レコード番号として、
R(I, 0)' 数値化したタグデータ
R(I, 1) 実レコード番号
430 ソートサブ（行1600）を呼ぶ
440 ■メニュー
480 メニュー選択サブ（行1800）を呼ぶ
490 カーソル位置に応じて各行へ飛ぶ
500 ■新規
510 ソートが必要ならソートサブを呼ぶ
R(0, 0) ソート要求フラグ（1のときソート要求）
520 登録する仮想レコード番号Vの更新
V 仮想レコード番号（配列Rの第1添字）
R(0, 1) 使用レコード数（有効な仮想レコード番号の最大値）
530 配列T$を消去／再宣言
T$ 単語データ処理用文字配列（バッファ用文字配列に添字で対応）
540 単語表示・入力サブ（行1400）を呼ぶ

```
560 Y=11:X=0:GOSUB 1800
570 ON X+1 GOTO 580,620,640
580 '□登録
590 IF V>N THEN R(V,1)=V:N=N+1
600 GOSUB 1300:GOSUB 1700
610 GOTO 520
620 '□訂正
630 GOTO 540
640 '□取消
650 V=V-1:R(0,1)=V
660 GOTO 440
670 '*** □めくる ********************
680 IF R(0,0) THEN GOSUB 1600
690 COLOR 1,15:CALL CLS
700 IF N=0 THEN 440
710 PRINT "めくり方は？":PRINT
720 PRINT "□ABC  □選択   □乱数"
730 Y=2:X=0:GOSUB 1800
740 ON X+1 GOTO 750,840,900
750 '*** □ABC
760 ST=1
770 '（順に見せる）
780 FOR I=ST TO R(0,1):V=I
790   GOSUB 1200        'データ読み出し
800   GOSUB 1400            '表示・入力
810   IF CH THEN GOSUB 1000   '変更選択
820   GOSUB 1700        '継続・中断選択
830 NEXT:GOTO 440
840 '*** □選択
850 IF INKEY$<>"" THEN 850
860 COLOR 15,4:CALL CLS
   :INPUT "先頭の2文字（半角）";T$
870 T=0:GOSUB 1500
880 FOR I=1 TO R(0,1)
     :IF R(I,0)>=T THEN ST=I:GOTO 770
890 NEXT:GOTO 440
900 '*** □乱数
910 V=RND(1)*R(0,1)+1
920 GOSUB 1200 'データ読み出し
930 GOSUB 1400 '表示・入力
940 GOSUB 1700 '継続・中断選択
950 GOTO 900
1000 '+++++ データ変更選択サブ +++++++
1010 LOCATE ,11
    :PRINT"□不変   □変更    □抹消";
1020 Y=11:X=0:GOSUB 1800
1030 ON X+1 GOTO 1040,1060,1090
1040 '+++ □不変
1050 RETURN
1060 '+++ □変更
1070 GOSUB 1300
1080 RETURN
1090 '+++ □抹消
1100 R(V,0)=&H7FFF
1110 LSET W$(0)=MKI$(R(V,0))
1120 PUT #1,R(V,1)
1130 R(0,0)=1
1140 RETURN
1200 '+++ データ読み出しサブ +++++++++
1210 GET #1,R(V,1)
1220 FOR G=0 TO 4
      :T$(G)=W$(G)
    :NEXT
1230 RETURN
1300 '+++ データ書き込みサブ +++++++++
1310 T$=T$(1):T=0:GOSUB 1500
    :R(V,0)=T
    :T$(0)=MKI$(T)        '+++ タグ設定
1320 IF R(V-1,0)>T OR R(V+1,0)<T
      THEN R(0,0)=1
1330 FOR G=0 TO 4
      :LSET W$(G)=T$(G)
    :NEXT
```

560　メニュー選択サブ(行1800)を呼ぶ
570　カーソル位置に応じて各行へ飛ぶ
580　「登録」
590　仮想レコード番号が総レコード数よりも大きいときは、その仮想レコード番号に対応する実レコード番号にVを代入、総レコード数を1増やす
600〜610　データ書き込みサブ(行1300)呼び出し／続行・中断選択サブ(行1700)呼び出し／行520へ飛ぶ
620〜630　「訂正」／行540へ飛ぶ(それまで入力していたデータを表示しながら再入力)
640〜660　「取消」／仮想レコード番号Vと使用レコード数R(0, 1)をもとにもどす／行440(メニュー)へ飛ぶ
670　■めくる
680　ソートが必要ならソートサブを呼ぶ
700　単語がなければ行440(メニュー)へ
710〜740　めくり方を選ぶ
750　「ABC」(アルファベット順最初から)
760　初期値STの初期化
770〜830　順にデータの読み出し／表示・入力／データに変更があれば、書き換えるかどうかのウインドウを出す／継続・中断の選択／ループ終了後行440(メニュー)へ
CH　表示・入力中に文字の変更があると−1に設定されるフラグ。
840　「選択」(単語の検索)
850　キーバッファクリア
860　最初の2文字の入力
T$　(単純文字変数)　タグ計算用文字列
870　タグ計算サブ(行1500)を呼ぶ
T　タグの値
880　指定された文字列(T$)のタグ値(T)を比べて単語の探索
890　行440(メニュー)へ
900〜950　「乱数」／乱数でVを設定し、表示(「乱数」では、行1000を呼び出していないので単語の修正などはできない)
サブルーチン群
1000　■データ変更選択サブ
1010〜1030　データ変更メニュー選択
1040〜1050　「不変」(データはオリジナルのまま)／そのまま呼び出し先にもどる
1060〜1080　「変更」／行1300(データ書き込みサブ)を呼び出す／もどる
1090〜1140　「抹消」／仮想レコード番号Vのタグを&H7FFFにする／ファイルに書き込む／ソート要求フラグを1にする／もどる
1200　■データ読み出しサブ
1210　仮想レコード番号Vを読み出す(配列RがVを実レコード番号に変換している)
1220　バッファ変数W$の内容を処理用配列T$に移す
1230　もどる
1300　■データ書き込みサブ
1310　タグ計算用文字変数T$に単語の綴りを入れる／タグ値Tを初期化／行1500(タグ計算サブ)を呼ぶ／配列RとT$のタグ設定
1320　そのタグを変更することで仮想レコードの順に変化が生じるかどうか／もし変化があるならソート要求フラグを1にする
1330　バッファW$に配列T$の内容を移す
1340〜1350　ファイルに書き込む／もどる

```
1340 PUT #1,R(V,1)
1350 RETURN
1400 '+++ 単語表示・入力サブ +++++++++
1410 COLOR 15,4:CALL CLS
1420 CH=0
1430 IF INKEY$<>"" THEN 1430
1440 FOR G=1 TO 4
      :LOCATE 0,G*2-2:COLOR 3
      :PRINT M$(G):PRINT T$(G);
      :COLOR 15:LOCATE 0,G*2-1
1450   LINEINPUT T$
      :IF G=1 AND T$="" THEN 640
1460   T$=LEFT$(T$+SPACE$(L(G)),L(G))
       :CH=CH OR (T$<>W$(G))
       :T$(G)=T$
     :NEXT
1470 T$=T$(1):T=0:GOSUB 1500
    :T$(0)=MKI$(T)
1480 RETURN
1500 '+++ タグ計算サブ +++++++++++++++
1510 FOR G=1 TO 2
      :A=ASC(MID$(T$,G,1)+" ") AND &H7F
      :A=A+(A>&H60 AND A<&H7B)*&H20
      :T=T+A*&H100^(2-G)
    :NEXT
1520 RETURN
1600 '+++ 並べ替え（ソート） +++++++++
1610 COLOR 8,15:CALL CLS
    :PRINT "単語を並べ替えています"
1620 FOR G=1 TO N-1
      :FOR H=G+1 TO N
        :IF R(G,0)>R(H,0)
         THEN SWAP R(G,0),R(H,0)
             :SWAP R(G,1),R(H,1)
1630  :NEXT
    :NEXT
1640 FOR G=N TO 1 STEP -1
      :IF R(G,0)=&H7FFF THEN NEXT
1650 R(0,1)=G:R(0,0)=0
1660 RETURN
1700 '+++ 続行・中断選択サブ +++++++++
1710 LOCATE 0,11,0:PRINT "ＳＰＡＣＥ=続
行　ＥＳＣ=中断";
1720 IF STRIG(0) THEN 1750
1730 IF FNESC THEN RETURN 440
1740 GOTO 1720
1750 IF STRIG(0) THEN 1750 ELSE RETURN
1800 '+++ STICK,STRIG 入力サブ +++++++
1810 GOTO 1840
1820 S=STICK(0):IF S=0 THEN 1850
1830 X=X+(S=3)*(X<2)-(S=7)*(X>0)
1840 COLOR 8:LOCATE X*10,Y,1:COLOR 15
1850 TIME=0:FOR W=0 TO 4:W=TIME:NEXT
1860 IF NOT STRIG(0) THEN 1820
1870 IF STRIG(0) THEN 1870
1880 LOCATE ,,0:RETURN
1900 '+++ [CTRL+STOP] 割り込みサブ +++
1910 LOCATE ,,0:COLOR 15,4
1920 IF INKEY$<>"" THEN 1920
1930 CLOSE
1935 IF RD THEN COPY F$ TO "A:"
    :CALL CHDRV("A:")
1940 STOP OFF:ON ERROR GOTO 0
1950 END
2000 '+++ エラー割り込みサブ +++++++++
2010 COLOR 8:PRINT "エラー発生！":COLOR
 10
2020 PRINT "↓＋ＳＰＡＣＥで 中断、エラ
ーメッセージＳＰＡＣＥでディスクを確認
してもう一度"
2030 IF NOT STRIG(0) THEN 2030
2040 IF STICK(0)=5 THEN 1900
2050 IF STRIG(0) THEN 2050 ELSE RESUME
2100 '+++ 空きレコードがない +++++++
2110 CALL CLS:COLOR 8
2120 PRINT "空きレコードがありません"
2130 TIME=0:FOR W=0 TO 100:W=TIME:NEXT
2140 RETURN
```

1400　■単語表示・入力サブ
1420　文字変更チェックフラグCHを初期化
1430　キーバッファクリア
1440　メッセージ(綴り、品詞などのデータ入力時の見出し)および配列T$の内容を表示
1450　入力受け付け／新規の「綴り」入力時にリターンキーのみ行640(取消)へ飛ぶ
1460　入力文字列を各項目の長さにそろえる／文字変更の有無をチェック(CH)／処理用配列T$に一時文字変数T$の内容を移す
1470〜1480　綴りT$(1)のタグ値を計算し、タグT$(0)に設定／もどる
1500　■タグ計算サブ
1510〜1520　変数T$の1文字目と2文字目を大文字に直したうえでキャタラクコードに変換／タグ値Tに〈1文字目のキャラクタコード〉×256+〈2文字目のキャラクタコード〉を入れる／もどる
1600　■並べ替え(ソート)
1620〜1630　仮想レコード番号の小さい単語のタグが大きい単語のタグよりも大きければ、配列Rの2要素を入れ換える(これをくりかえすとアルファベット順にならぶ)
1640〜1660　ソートを終了すると「抹消」されたレコードが最後のほうにたまっていることを利用して、現に有効なレコード群の最大仮想レコード番号を検索／その番号を最新の使用レコード数R(0, 1)に設定／ソート要求フラグを初期化／もどる
1700　■続行・中断選択サブ
1720　スペースキーが押されれば行1750へ
1730　ESCキーが押されれば行440へ
FNESC　ユーザー定義関数(行180で定義)。ESCキーに関してSTRIG変数と同様の働き
1740　行1720に飛ぶ(くりかえし)
1750　スペースキーが押されたままならおなじ行で待ち、放されたら呼び出し先にもどる
1800　■STICK、STRIG入力サブ
1810　行1840へ飛ぶ
1820　変数Sにカーソルキーの押された方向を入れる(押されていなければ行1850へ飛ぶ)
1830〜1840　カーソル座標Xの計算／カーソル表示
1860　スペースキーが押されなければ行1820へ
1870　スペースキーが押されたままなら待ち
1880　カーソルを不表示にしてもどる
1900　■CTRL+STOP割り込みサブ
1910　カーソルと画面色を初期状態にもどす
1920　キーバッファクリア
1930　ファイルを閉じる
1935　現在RAMディスク上にあるものをAドライブにコピー
1940〜1950　ストップキー割りこみ停止／エラー割り込み解除／プログラム終了
2000　■エラー割り込みサブ
2010〜2020　操作メッセージ表示
2030　スペースキーが押されなければ待ち
2040　カーソルキー下が押されていれば行1900(CTRL+STOP割り込みサブ)に飛ぶ
2050　スペースキーが押されたままなら待ち、放されたらエラーを取り消してもどる
2100　■空きレコードがない
2110〜2140　メッセージを表示してもどる

BASICテクニック

付録ディスク
→P92の「スーパー付録ディスクの使い方」参照

知っている人は知っているが、知らない人はぜんぜん知らなかったりするCIRCLE文の円グラフ用の機能。新しい選挙制度に思いをはせつつ、円グラフをかいてみよう。

円グラフをかくための基礎知識

去年の年末から今年のはじめにかけて、まるでスポーツの話題のように国会で起きていることが報道されていた。国会の各派の勢力が微妙なバランスでできているからこそ、こんなふうにエキサイティング国会になるのだろう。論議されているものの中身は、恥ずかしながらなにがそんなにちがうのか、よくわからないままだが、日本新党、さきがけ、新生党などの勢力が勝ったということだけはわかる。

実際の政治的状況は衆議院の議席が表しているはずだが、どういう分布になっていただろう。

データを視覚的に見る方法はいくつかあるが、ある集合のなかでどの勢力がどのくらいの数を占めているかを見るのなら、円グラフしかない。

円グラフといえば、CIRCLE文には、この日のために円グラフ用の機能が備えつけられている。日本の政治の行く末を案じつつ、国会(衆議院)の議席を円グラフにしてみよう。

■一般的な円グラフについて

円グラフを作るときは、まず360度に対応する全体数を計算する必要がある。この場合は、合計551。ちなみにこれは衆議院の定数で欠員なしの状態だ。

この全体数から、各派が占める割合を算出する。たとえば、自民の場合、

222÷551≒0.403

円グラフにおける角度に変換すると、

360×0.403=145.08

円の中心点から垂直に上に線をかき、145度右に回転したところにまた線をかけば、自民党の勢力が円グラフ上に表される。

コンパスと分度器と定規を使ってかく円グラフなら、話はここでおしまいだ。

しかし、MSXのCIRCLE文を使うとなると、いくつかめんどうな問題がある。

■CIRCLE文による扇形

その問題のことはちょっと置いて、CIRCLE文の円グラフ用の機能のことについていちおう紹介しておこう。

CIRCLE文は、とうぜん円をかく命令だが、開始角と終了角を指定することで部分円をかくこともできる。その開始角、終了角の値に"ー"(マイナス)を付けると、その角度の半径がラインとして引かれるのだ。どちらもマイナス記号をつけておくと、扇形をかくことができる。これだけだが、これが円グラフ用でなければ、いったいなんのための機能だろうか。

さっき計算したような各党の勢力を円グラフでかくときの角度と、以上のCIRCLE文の機能があれば、かんたんに円グラフがかける……はずだが、そうかんたんにはいかない。

■頭を混乱させる2つの特性

まず、かんたんな問題から。

CIRCLE文では、角度を度数ではなく、ラジアンで指定する。ラジアンとは、360度を2πとして指定する方法だ。

これは、その党派の占める割合に2πをかければかんたんにラジアン指定の角度が出てくるわけだが、そのまえにπの正確な値を与えてやらなければならない。πの値は、以前から何度も使っているが、ATN(逆正弦)という関数から正確な値を仕入れることができる。プログラムのはじめに、この関数を使って、πの正確な値を持つ変数を作っておけばいい。

また、角度の値として使用可能な値は絶対値で0～2πの範囲内だという点も要注意だ。

しかし、次のような、CIRCLE文の2つの特性にはずいぶん混乱させられた。

①CIRCLE文の円は、左回

シュウキ゛イン

222	シ゛ミン
76	シャカイ
61	シンセイ
53	サキカ゛ケ・シントウ
52	コウメイ
19	ミンシャ
15	キョウサン
8	ソノタ

現在の衆議院の勢力図

◉図1　CIRCLE文の角度について

(a)開始角、終了角、円をかく方向

B2

B1

CIRCLE(X,Y),R,C,B1,B2,P

開始角　終了角

(b)角度のはかり方

$\frac{\pi}{2}$

π

0.2π

$\frac{3}{2}\pi$

◉図2　強調表示の計算

りにかかれる(図1左)
②CIRCLE文の角度は、中心点からXの正の方向に進んだ線を基準にしている(図1右)

これをふつうに円グラフをかくときと比べると、①は正反対、②は直角ぶんずれている。

これに対応して、次のような対策にたどりついた。
①扇形をかく目的の角度を開始角、逆に出発する角度を終了角として扱うこと
②2分のπの角度から開始すること(プログラムの行350で設定しているB1)

これによって、見た目どおりの円グラフをCIRCLE文でかくことができる。

■ついでにできる強調表示

また、ついでにある勢力だけを強調してみたいときのための機能も考えてみた。強調のしかたはいろいろあるだろうが、ここではその部分だけを飛び出させるという表現をとった。

これは意外とかんたんで、円をかくときの中心点を、円の外側にずらしてやればいいだけだ。どのようにずらせばいいかというと、開始角と終了角のちょうど真ん中の角度で、適度な距離だ(図2)。この角度は、PAINT文で色を塗る場合の点の座標を割り出すためにも使う。

＊

そして、EN-GRAPH.BP4ができあがった。行1000以降のデータを書き換えれば、いろいろな円グラフがかける。たとえば、1日の自分の行動の内訳を円グラフにしてみると、自分がいかに惰眠をむさぼり、だらだらと食事をし、遊んでばかりいるかということがよくわかる。

◉強調機能付き円グラフ表示プログラム EN-GRAPH. BP4 MSX 2/2+ VRAM64K

```
100 '===ｶﾞﾒﾝ ｼｮｷｶ
110 SCREEN 5,0:COLOR 15,0,0:CLS
120 OPEN "GRP:" AS #1
130 '===ﾍﾝｽｳ ｼｮｷｶ
140 PI=ATN(1)*4 '===ｴﾝｼｭｳﾘﾂ
150 X=70:Y=100:R=50:ES=20:L=16:P=1.1
160 LX=X+R+5:Q=R*.(0
170 '===ﾃﾞｰﾀﾖﾐｺﾐ
180 READ N        '===ｸﾞﾗﾌｺｳﾓｸ ｿｳｽｳ
190 DIM N$(N),A(N),E(N),B(N)
200 READ N$(0)   '===ｸﾞﾗﾌ ﾉ ﾅﾏｴ
210 FOR I=1 TO N
220   READ N$(I),A(I),E$
230   E(I)=VAL("&H"+E$)
240   A(0)=A(0)+A(I)
250 NEXT
260 '===ｿｰﾄ
270 FOR I=1 TO N-1
280   FOR J=I+1 TO N
290   IF A(I)<A(J)
        THEN SWAP A(I),A(J)
            :SWAP N$(I),N$(J)
            :SWAP E(I),E(J)
300   NEXT
310 NEXT
320 '===ｸﾞﾗﾌ ﾉ ﾅﾏｴ ｦ ｶｸ
330 DRAW "BM=X;,10":PRINT #1,N$(0)
340 '===ｸﾞﾗﾌ ｦ ｶｸ
350 B1=PI/2      '===ｻｲｼｮﾉ ｶｲｼ ｶｸﾄﾞ
360 FOR I=1 TO N
370   '===ｶｸﾄﾞ ﾉ ｹｲｻﾝ
380   B(I)=A(I)/A(0)*PI*2
390   B2=B1:B1=B1-B(I)
400   IF B1<=0 THEN B1=B1+2*PI
410   '===ｲﾛ ﾉ ｹｲｻﾝ
420   C=E(I) AND 15
430   '===ｷｮｳﾁｮｳ､ﾍﾟｲﾝﾄｽﾙ ﾃﾝ ﾉ ｹｲｻﾝ
440   D=B1+B(I)/2
450   DX=COS(D):DY=-SIN(D)
460   IF E(I)¥16
        THEN EX=ES*DX:EY=ES*DY*P
        ELSE EX=0:EY=0
470   '===ｸﾞﾗﾌ ﾄ ﾓｼﾞ ｦ ｶｸ
480   CIRCLE (X+EX,Y+EY),R,C,-B1,-B2,P
490   PAINT STEP(Q*DX,Q*DY*P),C,C
500   LINE (LX,(I+1)*L)-STEP(8,8),C,BF
510   DRAW"BM+4,-8"
    :PRINT #1, USING"### &          &"
                    ;A(I);N$(I);
520 NEXT
530 GOTO 530
1000 '===ﾃﾞｰﾀ
1010 DATA 8,ｼｭｳｷﾞｲﾝ
1020 DATA ｼｬｶｲ          ,76  ,09
1030 DATA ｼﾝｾｲ          ,61  ,07
1040 DATA ｻｷｶﾞｹ･ｼﾝﾄｳ,53  ,1A
1050 DATA ｺｳﾒｲ          ,52  ,0D
1060 DATA ﾐﾝｼｬ          ,19  ,05
1070 DATA ｷｮｳｻﾝ         ,15  ,08
1080 DATA ｼﾞﾐﾝ          ,222 ,04
1090 DATA ｿﾉﾀ           ,8   ,0E
```

■画面初期化
110　スクリーン設定、色設定
120　グラフィック画面に文字を表示する準備

■変数初期化
140　π(円周率)計算
　PI　円周率
150〜160　円描画用変数の設定
　X、Y　円の中心座標
　R　円の半径
　ES　強調表示のときの離れぐあい
　L　表示される文字の行送り
　P　偏平率(真円に見えるよう調整)
　LX　色見本の四角を表示するX座標
　Q　中心からペイントする点までの距離

■データ読みこみ
180　データの項目数読みこみ
190　それに応じた配列宣言
　N$(n)　項目名／N$(0)はグラフの名前に使用
　A(n)　項目nの素データ／A(0)は全体数として使用
　E(n)　項目nの特性(強調表示の有無と色コード)　※E$から変換
　B(n)　項目nが占める角度
200　グラフの名前読みこみ
210〜250　各項目のデータ設定
　220　項目名、素データ、特性データ読みこみ
　230　特性データE$をE(n)に変換
　240　全体数計算

■ソート
270〜310　素データの大きい項目ほど添字が小さくなるようにソート(2数を比べ、添字の大きいほうの素データが小さければ名前ごと入れ換える)
320〜330　グラフの名前をかく

■グラフをかく
350　見た目の角度0を2分のπとしてB1に設定
　B1　CIRCLE文の開始角
360〜520　各項目を円グラフでかく
　370〜380　その項目が占める角度B(n)を計算
　390　終了角B2に出発する角B1を設定、開始角B1に目的の角度を設定
　B2　CIRCLE文の終了角
　400　B1が0になったときのみ、B1を−2πとする　※0ではマイナスと見なされず、扇形にならないため
　410〜420　E(n)からCを計算
　C　項目の色
　430〜470　強調、ペイント用の計算
　D　開始角と終了角の中間の角
　DX、DY　角度Dのsinとcos
　EX、EY　強調表示用の座標増分
　480　グラフをかく
　490　各項目に決められたの色を塗る
　500　項目名用色見本の四角をかく
　510　色見本の四角から4ドットはなして、項目名と素データを表示する
530　永久ループ

■データ
1010　項目数とグラフの名前
1020〜1090　それぞれ、項目名／素データ／特性データ(16進数2桁で表現。上位1桁が0でなければ強調表示、下位1桁は色コード)
※データを変更して別の円グラフをかくときは、行1010の項目数とグラフの名前を変更し、その項目数とおなじ数のデータを上記の順に設置してください。

※国会のなかの会派は、厳密にはかならずしも政党と一致するわけではありませんが、ここでは簡便化のため、よく知られている政党の名前をそれらの代表として表示しました。

BASICピクニック

付録ディスク

→P92の「スーパー付録ディスクの使い方」参照

ポリゴンを使った立体的表現のゲームがもうあたりまえになりつつある。そこにステレオグラムの手法を組み合わせれば、裸眼で飛び出して見えるゲームができるかもしれない。

動くステレオグラム

世が世なら、MSX3はあんなふうになるはずだったのでは……と思ってしまう3DOソフトのラインアップを見ていると、もうそこではポリゴンによる3D感覚のゲームがすでに1つのジャンルのようになっている。

ポリゴン(polygon)とは、ほんらい多角形のことだが、ポリゴンゲームといった場合、その多角形の面を組み合わせて立体物を表現するプログラミング技法のことを指す。よく学校の美術室にゴツゴツした感じのデッサン用石膏像(面取り)が置いてあるが、ようするにあれがポリゴンだ。そして、その石膏像どうしが荒野で戦うと『バーチャファイター』になる。

◆

立体物をポリゴンで表しておくと、人間の顔などは見た目が気持ち悪くなるという欠点(アローン・イン・ザ・ダーク現象)があるものの、わずかなデータで立体物を表すことができるという決定的な利点がある。

へたをするとドットごとにデータをとっていかねばならないところを、それこそ点と線で管理することができるのだ。

だから、さまざまな計算によって立体物をコンピュータ上で回転させたり、変形させたりすることも比較的かんたんだ。

安くて高性能のパソコンやゲーム機が現れてきたからには、これから2、3年は、ポリゴンゲームの時代になるだろう。かつて、粗いドット絵がテレビゲームの象徴だったのとまったくおなじ理由で、ゴツゴツしたポリゴンCGが今後数年のゲームの象徴になるにちがいない。

しかし、そのポリゴンもせいぜい「3Dぽく見える」というだけで、ほんとうに飛び出して見えるのではない。

どうせなら、ほんとうに飛び出して見えるゲームができないだろうか。

できる――はずだ。

◆

図の(a)は、ピラミッド型の物体を左右の目が見ている状態の俯瞰図だ。このとき、(b)のように左右の目にはそれぞれ角度のちがう像が見えている。

逆にいうと角度のちがう像を左右の目に見せれば、人はそれを立体だと感じるはずだ。これがステレオグラム(立体画像)の原理だ。

ステレオグラムにはさまざまな方法があるが、もっとも手軽にできるのが、裸眼のまま像を重ねて見る方法で、いわゆる、交差法(寄り目：右目で左の絵、左目で右の絵)というやつだ。もう1つの平行法(遠い目：右目で右の絵、左目で左の絵)ではうまくいかない。

というのも、テレビ画面くらいの大きさで平行法をやろうとすると、平行どころか、視線を左右に広げなければならなくなるからだ。それができたら、あなたはカメレオンだ。

◆

交差法は、図の(c)のように、見える物体の仮想的な位置で左右の視線が交差し、その向こうに置いてある絵を見るようなかっこうになる。そういうわけで、図(d)のように左右の像を入れ換えてならべると、それを寄り目で見たときに立体的に見えるわけだ。ためしに、図(d)の左右の図を寄り目で重ねあわせて見てほしい。中央のピラミッドの稜線がこちらに浮き上がって見えるはずだ。

◆

この原理を応用して作ったサ

↑サンプルの「3D-GRAPH」の実行画面。2つの枠が重なるように寄り目で見るとダイアモンドが立体的に見える。さらに、カーソルキーの左右で立体的なままダイアモンドがまわる　→ページー(裏画面)にはあらかじめ全グラフィックがかきこまれている

■図　寄り目で見るとなぜ飛び出すのか

(a)三角錐を見る右目と左目(上から見た図)

左目
右目

(b)それぞれの目に見える像

左目の像　右目の像

(c)ステレオグラム(交差点)を見る右目と左目(上から見た図)

左目　A
右目　B

(d)寄り目で見ると飛び出して見えるステレオグラム

A(右目用)　B(左目用)

ンプルプログラムが付録ディスクに入っている「3D－GRAPH」だ。

左右の枠に表示されたダイアモンドのグラフィックは10度ずらした角度でかかれている。カーソルキーの左右を押すと、その差を保ったまま、押された方向に回転する。2つの画像を寄り目で重ね合わせて見ると、飛び出して見えるダイアモンドが左右にくるくる動くわけだ(左右それぞれに限界がある)。

左の枠も右の枠も、それぞれダイアモンドがぐるりと回転するシンプルなCGアニメーションだが、その2つを寄り目であわせるだけで幻のようにワイアフレームのダイアモンドが浮かびあがってくる。

プログラムは、ダイアモンドの3次元計算(この部分はこのコーナーでなんとか利用したもの)にほとんどをついやし、ページ1(裏画面)に回転するダイアモンドをコマ落とし(10度きざみ)でかいている。ページ1からページ0へ絵をコピーするときに、右目用と左目用の絵が10度ずれるようにすれば、ステレオグラムができあがる。そのずれを保ったまま、カーソルキーの操作にあわせてコピー元を変えていけば、動くステレオグラムができる。

これは、真の3Dゲームへの第1歩だ。とりあえず、裸眼で遊ぶ真の3Dゲームのゲーム画面は縦半分で、右側に左目用の画像、左側に右目用の画像を表示すればよいことがわかる。

あとは、角度のズレが大きくなれば近くに見え、小さくすれば遠くに見えるなどの細かな法則を研究してほしい。

寄り目でなければ遊べないゲームは、スーパーファミコン用は絶対に出ないが、ファンダムなら積極的に採用するだろう。ぜひがんばってほしい。

リストはスペースがないので掲載していないが、SCREEN0のWIDTH40の状態で見やすいように作ってあるので、興味のある人は画面上で確認してほしい(解説は右のカコミ)。

「3D－GRAPH」プログラム解説　3D－GRAPH. BP6

MSX2/2+ VRAM128K

■各種設定

1020　文字列領域確保／変数の型宣言〈スクリーン／色〉

1040　スクリーン・色初期設定

1050　パレットコード4および1、3、7の色をパレット定義⇨背景色設定とダイアモンド表示のさいに奥行き感を出すためのグレー3種作り

1060　ページ1(裏画面)をパレットコード4でぬりつぶす

1070　ページ0にもどる

〈描画用設定〉

1090～1140　変数設定、配列宣言

R　画面縦横比率調整用(このままで円をかくと縦長の楕円に見えるため)

U　10度をラジアンに変換した数値

T(n, m)　回転による座標値の変化を計算するための係数

N　立体の頂点の数－1

M　辺の数－1

OX、OY　表示するときの原点となる座標(じっさいにはこの値を増減して表示している)

X(n)、Y(n)、Z(n)　頂点nのXYZ座標(3次元座標)

XX(n)、YY(n)　頂点nに対応する画面上の表示座標(値の計算は行2120で)

A(n)、B(n)　辺nの両端にある頂点番号

1150～1160　頂点座標読みこみ(読みこんだあと素データを一律2分の1に縮小)／XM、YM、ZM計算

XM、YM、ZM　すべての頂点のX、Y、Z座標の平均値

1170　点(XM, YM, ZM)を原点として各頂点の座標再設定

1180　立体ダイアモンドをかくためのすべての辺のA(n)、B(n)を読みこみ

〈コピーするための設定〉

1200～1250　ページ0で10度ずつ回転したダイアモンドをかき、それを1つずつページ1にならべていって、動画用の素材を作る。そのときに使用するコピー先座標などのパラメータの設定。

ST　ページ1へのコピー開始フラグ(これが0のときはページ0にかくだけでページ1にはコピーしない)

C(n)　0～2を循環して使うための配列("MOD3"とおなじ)

CX、CY　4×4に仕切ったページ1の各ブロック番号(CX＝0、CY＝0だと左上のブロック)

SX、SY　コピーする1単位のグラフィックの大きさ

CX(n)、CY(m)　ページ1のブロック(n, m)のXY座標

LX、LY　(動画表示時)左に表示される画像(右目用画像)の基準座標

RX、RY　(動画表示時)右に表示される画像(左目用画像)の基準座標

■ダイアモンド描画

2010　回転データ(行4230～)読みこみ

A　ダイアモンドを描画するときに回転の有無、回転軸などを指示するデータ。Aの値と意味の対応は以下に示す。

0＝次の手順(ステレオ表示)へ
1、2、3＝それぞれX、Z、Y軸を中心に回転
4＝逆回転(逆回転の手続きだけ)
5＝ページ0からページ1への転送開始

2020～2040　読みこんだAの値に応じた処理

2050　Aの値から1を引いて次の回転計算へ

〈回転計算〉

2070～2110　全頂点について指定された回転の計算をほどこす

【参考】点(x,y,z)をX軸中心に角a回転すると、座標は、(x,y＊cos(a)－z＊sin(a), y＊sin(a)＋z＊cos(a))

2120　全頂点について表示する実座標を計算

W　奥行き係数。画面の奥にある点ほど原点の近くに見えるように調整するためのもの

2130　画面クリア

2140～2170　ダイアモンド表示(行2150でLを計算し、2160で立体ダイアモンドの1辺1辺を描画)

L　描画する辺のパレットコード(奥行き感を出すためのグレー3種および15の白を選択)

〈ページ0からページ1にコピー〉

2190～2240　ページ0にかかれたダイアモンドをページのブロックに順番にコピーしていく(4×4ブロックのすべてをかきおわったらSTを0にして行2010へもどる)

■ステレオグラム表示

3010～3050　初期設定／枠準備

3060　カーソルキー入力受け付け

S　スティック入力値

3070　ページ1からページ0にコピーするブロック番号P

P　ブロック番号(左上のブロックを0として右に1、2、3と数えていく)

3080～3090　変数Pの限界チェック

3100　画像番号からブロック座標計算

3110　右側に左目用画像をコピー

3120　右目用に変数Pを計算(左目用画像よりも1段階回転したもの)

3130　左側に右目用画像をコピー

3140　行3060にもどる

■データ

4020　頂点の総数、辺の総数、原点

4040－4160　立体の3次元座標

4180－4210　すべての辺の両端の頂点番号　2つずつでセットになっている)

4230　ダイアモンドの回転シナリオ

BASICピクニック

→P92の「スーパー付録ディスクの使い方」参照

コンピュータを持っているからには人工頭脳としてのコンピュータに出会いたい。MSXを人工頭脳化するために、最初のごくわずかな一歩を踏み出そう。

コンピュータの「思考」

コンピュータは「思考」するか

さすがに最近はあまり見かけなくなったが、かつてはコンピュータの出力がちょっと変だったりすると、「コンピュータもたまにはミスをするんです」などといいわけする人々がいた。それは人間のミスをごまかすためのいいわけかもしれないが、ついコンピュータに人格を感じてしまうからでもあるだろう。

コンピュータには、基本的に「考えている」という感じがある。心が純粋なころは、
「ただいまコンピュータが考えています」
といわれれば、ことばどおりに受け取って、あれこれ思い悩むコンピュータの姿を思い浮かべていたものだった。

コンピュータははたして「考える」ものだろうか。

人間の設計した回路の組み合わせのなかで、人間の組んだプログラムが実行されているだけではないか。もちろんそのとおり。しかし、それでは、人間は「考える」ものだろうか。シナプスによって複雑に連結された脳細胞のなかを微弱電流が流れているだけではないか。

730万個の破壊された脳細胞

ほんとうかどうか知らないが、たばこを1本すうたびに脳細胞が少なくとも100個破壊されるという。1日20本吸う人は珍しくないし、そのペースで10年間たばこを吸いつづけている人も珍しくない。ということは、
100×20×365×10=7300000

脳細胞が730万個破壊されている人も珍しくないということになる。人間の脳は、そのていどのことではなんともないほど大きく複雑なのだ。シナプスを流れる微弱電流にすぎないものも、大きく複雑なシステムに組みあげられると精妙な意思となって現れる。

思考の本質のごく一部

つまり、量の問題なのだ。

コンピュータの思考は、あまりにも小さな記憶空間とあまりにも単純な回路のために、人間の柔軟な思考の足下にも及ばないが、しかしそれでも、思考の本質のごく一部を写している。

思考の本質のごく一部とは、①状況を認識し、②認識に応じて行動する、の2点だ。

その意味で「思考」するプログラムを作ってみよう。

むずかしいものはいずれにしてもできないので、ほんの試し斬りていどの意味で8パズルでやってみた。

「8PUZ-SOL.BP8」の使い方

先にできたものを見せてしまおう。付録ディスクの「8PUZ-SOL.BP8」というファイルだ。

実行すると、8パズルの最終形とメッセージが表示され(このページ左下の写真)、人間側が問題を作るところから始まる。

カーソルキー(またはジョイパッド)でピースを移動させてならべかえていく。あたりまえだが、動かせるピースはブランク(数字のないマス)と隣り合わせのピースにかぎる。

十分かきまぜたら、スペースキー(またはAボタン)を押すとコンピュータが考えはじめる。

かなり意地悪くかきまぜたつもりでも、数秒で解いてしまう。その数秒も、ピースを動かすときに約6分の1秒のウェイトをかけている(行2140)からで、それをとりはずすとあっというまに解いてしまう。

「ROTATE」の発見

どうやってパズルを解くか。

人間がピースをならべかえていく手順をすべて記憶しておき、それを逆にたどる……というの

プログラム実行直後の画面。まず、カーソルキーでピースをならべかえる

てきとうなところで、スペースキーを押す。すぐに、コンピュータがこのパズルを解きはじめる

ならべかえ中。いま、6を定位置に移動させるために4と5をいったん退避させている場面

数秒後に終了。かんたんに解かれてくやしいときは、スペースキーを押してもういちど

も1つの解法だろうが、もちろんそんなことはしていない。

また、8パズルがとりうるすべてのならびとそれぞれに対応した解法の手順を記憶しておき、現在のならびを判定したうえで適用するのも1つの解法だが、それは不可能だ。

ここでは、「1」から数字順に地道に場合分けしながら定位置に持っていくようにしている。

が、ピース単位で場合や手順を考えていると、頭がパンクしてしまった。思考の経済のために複数の手順を組み合わせた複合手順をまず探した結果、シンプルで便利な複合手順を見つけた(図1)。これは、ブランクを含む2×2の範囲内で、ブランクの位置を変えず、ピースの順番を1つずつ回転するようにずらす。右回りも左回りもできる。この簡素な複合手順をとりあえず「ROTATE」と名付けた。

コンピュータがパズルを解きはじめる最初の部分(プログラムの行1020〜1050)で、ブランクを中央に持ってきているが、それはこのROTATEを使いやすくするためだ。

具体的には「1」を定位置まで持ってくる手順は、すべてROTATEの組み合わせだし、たとえば図2のように、ピースAの下にあるピースBを、Aの右横に持ってくる手順もROTATEの組み合わせでできる。

思考ルーチンの基本戦略

パズルを解く思考ルーチンを組み上げていく基本戦略は、
①動かしたいピースが現在どこにあるかによって場合分けする
②ROTATEの組み合わせで手順を考える
③それでもだめなら、1ピース単位の移動を考える
の3つによった。

この戦略にのっとって、数字順にピースを移動させる手順を積み重ねていくと、このプログラムになるというわけだ。

プログラムを見返すと、IF文(場合分け)とGOSUB文(ROTATE、1マス移動、特定のピースの場所を調べるサブルーチン)の連続だが、考えてみれば、人間の思考だって、大筋、こんなものではないか。ケースごとに、過去に成功した対処方法をもっていて、ケースに照らし合わせ、そのサブルーチンを呼ぶ。ある種の人々は、ある年齢から先はそれだけで生きているフシがある。フェイドアウト。

●図1 "ROTATE"の手順

たんに、B、A、Cの順にピースをブランクのほうに移動させているだけなのだが、一巡すると、ブランクの位置はもとのままで、3つのピースの位置が右回りに1つずつずれているという仕組み。

●図2 Aのとなりに Bを持ってくる手順

●「8 PUZZLE SOLVER」の解説

8PUZ-SOL. BP8 MSX2/2+ VRAM64K

110 ■初期設定
120〜130 画面初期設定/グラフィック画面に文字を表示する準備
140〜150 変数初期設定
OX、OY:パズルの表示基準座標
GX、GY:「ROTATE」される範囲の左上マスの仮想座標 ※以下、「座標」はすべて仮想座標
R:「ROTATE」の方向。1=左回り、−1=右回り
160〜170 配列設定
X(n)、Y(m):ピース表示の実座標 ※(n, m)がピースの座標
V(n, m):座標(n, m)にあるピースの数字(ブランクは9)
RX(n)、RY(n):「ROTATE」するピースの座標。nは0〜3で左上のピースから右回りに対応
180〜200 ユーザー定義関数定義
FNR(n):0〜3の繰り返し
FNS:方向入力(カーソルキーまたは十字ボタン)受け付け
FNT:トリガー入力(スペースキーまたはAボタン)受け付け
210 ピースが動くときの音設定
220 ■パズルのピース作成
230〜320 (OX、OY)を基準点として、16×16ドットのピースと数字を表示し、ピースが動く音を鳴らす
X、Y:ピースの座標
N:ピースに書きこまれる数字
330 ■ピースをかきまぜる
340 メッセージ表示
350〜370 キー入力(カーソルキー、スペースキー)受け付け/入力待ち/スペースキーが押されたら行1000へ
380 ブランクの現在の座標をチェック(行2000呼び出し)
BX、BY:ブランク専用の座標変数
390〜410 ブランクから見てカーソルキーを押した方向にあるピースの座標をX、Yに入れる/パズルの基本枠からはずれていれば行350へ
420〜430 ピースをブランクに移動する(行2100呼び出し)/行350へ
1000 ■パズル解き開始
1010 最初のメッセージをぬりつぶす
1020 ■ブランクを中央に動かす
1030〜1050 ブランクの座標をチェック/X、Yの順で中央に寄せる
1100 ■「1」を動かす
1110 「1」の座標チェック
1120 (0,0)なら行1200へ
1130 (0,1)なら(0,0)〜(1,1)の範囲で右回り、(1,0)なら同様に左回りの「ROTATE」を実行(サブ呼び出しは行1170で。以下同)
1140 (0,2)なら(0,1)〜(1,2)の範囲で右回り、(2,0)なら(1,0)〜(2,1)の範囲で左回りの「ROTATE」を実行
1150 (1,2)なら(0,1)〜(1,2)の範囲で左回り、(2,1)なら(1,1)〜(2,2)の範囲で右回りの「ROTATE」を実行
1160 (2,2)なら(1,1)〜(2,2)の範囲で左回りの「ROTATE」を実行
1170 行1130〜1160を受けて「ROTATE」(行2100)を呼び、行1110へ
1200 ■「2」を動かす
1210 「2」の座標チェック
1220 (1,0)なら行1300へ
1230 (2,0)か(2,1)なら(1,0)〜(2,1)の範囲で、Yが0のとき左回り、1のとき右回りの「ROTATE」を実行(サブ呼び出しは行1260で。以下同)
1240 (1,2)か(2,2)なら(1,1)〜(2,2)の範囲で、Xが1のとき右回り、2のとき左回りの「ROTATE」を実行
1250 (0,1)か(0,2)なら(0,1)〜(1,2)の範囲で、Yが1のとき右回り、2のとき左回りの「ROTATE」を実行
1260 行1230〜1250を受けて「ROTATE」(行2100)を呼び、行1210へ
1300 ■「3」を動かす
1310 「3」の座標チェック
1320 Y座標が0なら、行1400へ
1330 Y座標が1なら、(0,0)〜(1,1)の範囲で左回りの「ROTATE」を実行/それ以外は行1360へ ※「3」が(0,1)のときと(2,1)のときの2つの場合がある。(0,1)なら「ROTATE」によって、「1」「2」が左回りに1つずれ、「3」が(1,0)に移動したあと、行1350によって3つとも定位置に移動する。また、(2,1)なら、「1」「2」をいったんずらすだけになるが、行1340で「3」が(1,0)に移動するので、前者と同様になる
1340 X座標が2なら、(1,0)〜(2,1)の範囲で右回りの「ROTATE」を実行
1350 (2,1)、(2,0)、(1,0)、(0,0)、(0,1)、(1,1)にあるピースをこの順に動かし、行1310へ ※(0,0)〜(2,1)の6マスを右回りにずらす
1360〜1370 「3」が(0,2)なら(0,1)〜(1,2)の範囲で右回り、(1,2)ならおなじ範囲で左回り、(2,2)なら(1,1)〜(2,2)の範囲で左回りの「ROTATE」を実行/行1310へ
※行1360は不要(ゴミ)
1400〜1660 同様に「4」〜「6」の処理
1700 ■「8」を動かす
1710 (2,2)のピースを(2,1)に移動
1800 ■最後の確認
1810〜1820 「8」が定位置になければ無断で再実行してごまかす
※行1710の段階で「8」が期待される位置にないと計画が狂い、ここを実行する。しかし、そうはならないだろう。8パズルは、ランダムなならびからはじめると、行1710の段階で「8」が(2,2)にある系列と、かわりに「7」がある系列の2つがあるようだ。定位置は前者の系列なので、行1820は実行されずにおわるはずだ。別の系列は定位置と左右反対のパターン。ただ、担当者には証明する能力がない。このへん、くわしい人がいたら教えてほしい
2000 ■指定ピースの座標チェック
2010〜2030 変数Nの値を持つピースが見つかるまでサーチ(変数X、Yにそのピースの座標が入る)
2100 ■ピースをブランクへ動かす
2110〜2150 (X,Y)のピースをブランクへコピー/(X,Y)のマスをブランクに(ぬりつぶす)/効果音/6分の1秒待つ/配列Vの内容を交換
2160〜2170 ブランク専用座標変数BX、BY更新/呼び出し元へもどる
2200 ■「ROTATE」
2210 ブランクの位置をチェック
2220〜2260 指定された方向へピースをずらしていく/呼び出し元へもどる

BASICピクニック

ターボRのPCM音源機能をプログラムに生かすための、毎度ながら“ほんのささやかな”ワンステップ。はたして、紙に印刷されたプログラムはしゃべるか？

印刷された音声

自由自在に加工できるデジタル

アナログデータと比べて、デジタルデータが持つ最大のメリットはなにか？　それは、デジタル化されたデータが、劣化することなく、何度でも、移動したり、複製を作ったりでき、きわめて柔軟で加工しやすいという点だ。

デジタイズされた絵（CG）は、電話回線で送ることもできるし、フロッピーディスクに入れておくこともできるし、データを表す文字にすることもできる。そして、そのどれもがオリジナルとまったくおなじだ。というより、デジタルデータの世界では、オリジナルと複製を区別することができない。

しかも、圧縮したり、2つに切ってまたあとでくっつけたり、全体に明るくしたり、伸ばしたり縮めたり、赤っぽくしたり、一部だけ切り取ったり、さまざまな加工が自由自在にできる。

一方、アナログの絵は、2キロ先に届けるのさえ、たいへんな手間だ。だれかが届けてくれるとしても、折れ曲がらないように厚紙ではさんだり、雨が降っていればビニール袋に入れたりしなくてはいけない。コピーをとればとるたびに粗雑になるし、うっかりコーヒーでもこぼそうものなら取り返しがつかない。アナログの世界では、オリジナルがかけがえのないものなのだ。もちろん、そこがアナログのいいところでもあるわけだが。

孫ファクス

デジタルデータの世界ではコピーによって劣化したり、変質したりすることはないが、アナログからデジタルへ移るときにはかなりひどい変質が起こる。

身近な、いい例がファクスだ。ファクスされたものをふたたびファクスで送る（孫ファクス）と、かなりひどい状況になる。

ターボRに標準装備されているPCM機能のデジタイズには、図1のサンプリングレートのように0～3の4段階のグレードがあるが、最低ランクの「3」でデジタイズした音は、孫ファクスに似ている。

ようやく、本題に軟着陸した。今回のBASICピクニックでは、この「3」の孫ファクス的なデジタイズ音を使って、音を合成しようという、けっこう無謀なことを試みている。

サンプリングレートを最低ランクの「3」にしたのは、できる

■図1　RAM、VRAM、配列へのPCM録音と再生

RAMへ録音	CALL PCMREC (@&HB000, &HB1FF, 3, 127, 1)
VRAMへ録音	CALL PCMREC (@&HB000, &HB1FF, 3, 127, 1, S)
配列へ録音	CALL PCMREC (@A%, ⋯⋯⋯, 3, 127, 1)
RAMから再生	CALL PCMPLAY(@&HB000, &HB1FF, 3)
VRAMから再生	CALL PCMPLAY(@&HB000, &HB1FF, 3, S)
配列から再生	CALL PCMPLAY(@A%, ⋯⋯⋯, 3)

圧縮スイッチ※3

トリガーレベル※2

録音・再生開始番地

録音・再生終了番地

サンプリングレート※1

長さ（バイト数）。省略すると配列の中身すべてになる

※1　0～3。数値は大きいほどあらく、小さいほど細かい。ただし、細かくなるとデータ量が大きくなる。
※2　0～127。録音を開始するレベルの指定。0だと実行後、ただちに録音を開始する。
※3　0または1。1だと無音状態のデータを圧縮する。

だけ、扱うデータを小さくしたかったからだが、それでも、ギザギザの「あ」を鳴らすだけで、十数行ものデータが必要だということもわかった。

なかなか無謀な計画であったうえに、記録的な猛暑で、まったく猛暑わけないが、ディスク収録にまにあわなかった。26ページの長いリスト2を打ちこむか、次号のディスクまで待ってください(次号のディスクに空きがあれば)。

理想と現実

さて、今回の計画で目指した理想の状態は、当初「あ」〜「お」とそれぞれの子音のデータを持つことで、ローマ字で入力された文章をMSXが読む、というものだった。

しかし、さまざまな現実にはばまれ、最終的にはそれの孫コピークラスになってしまった。「あ」〜「お」、子音のk、s、tのデータを持つことで、ローマ字で入力された、あ行〜た行の20音を読む。

ほんとうは、「し」、「ち」、「つ」の子音はs、tではないのだが、このていどの音質ならs、tで代用しても変わらない。

作成の手順について

手順としては、まずリスト1のような(「のような」というより、まさにこれを使ったのだが)プログラムで、自分の声をデジタイズすることから始める。

リスト1を実行すると、マイクから音が入ってくるまで待機状態になるので、よけいな音をたてないように注意しながら、「あ〜」と発音してみる。

すると、デジタイズされた音を再生しながら、そのデータを行1000〜1030のDATA文として画面に表示するので、カーソルをそこに持っていって、リターンキーを押していくと、データが登録される。音ごとにデータをセーブしておいて、あとで行番号を変えながらつなぎ、リスト2のDATA文を作った。

子音は、「か」とか「す」でデジタイズしたあと、データの状態で分割して、子音部分だけを切り出した。……というと聞こえがいいが、基本的には前半を切り取っただけだ。

再生するときは、それぞれの音のデータをいったん配列に入れておき、入力された文字に応じて、メモリにデータをならべていく。メモリはとうぜんだが広くて安全なVRAMを使った。この部分のプログラムには、コピーに使う配列の基本的構造に関する知識が必要だが、これについては図を見てほしい。

トーキングロボットを目指して

「KASITE」「TASUKE-TE」「SUKISUKI」などのことばしかしゃべれないが、幼稚園に入ったばかりのインコを孫ファクスしたくらいのしゃべりは聞かせてくれる。もっとデジタイズに注意して、時間をかけてつくれば、トーキングロボットくらいはできるのではないだろうか。そして、そのロボットに、通信で遊ばれている人工無能のシステムを加えれば、話し相手ができて楽しそう……か?

■リスト1(録音用)

```
100 'record-voice
110 CLEAR 200,&HB000
120 CALL PCMREC(@&HB000,&HB0FF,3,127,1)
130 FOR I=0 TO 3
140   PRINT 1000+I*10;"DATA ";
150   FORJ=0 TO &H3F
160     AD=I*&H40+J
170     A =PEEK(AD+&HB000)
180     A$=RIGHT$("0"+HEX$(A),2)
190     PRINT A$;
200   NEXT
210   PRINT
220 NEXT
230 CALL PCMPLAY(@&HB000,&HB0FF,3)
```

■図2　画像データ用配列の構造

A%(0)…………X方向のドット数
A%(1)…………Y方向のドット数

A%(2)…………1バイト目のデータ
　　　　　+2バイト目のデータ*256
A%(3)…………3バイト目のデータ
　　　　　+4バイト目のデータ*256
A%(4)…………5バイト目のデータ
　　　　　+6バイト目のデータ*256
A%(5)…………7バイト目のデータ
　　　　　+8バイト目のデータ*256
⋮
A%(n)…………(n-1)*2-1バイト目のデータ
　　　　　+(n-1)*2バイト目のデータ
　　　　　*256

Mファンにいいたい放題!

★毎号読ませてもらっています。オールディーズ、CG、C言語、マシン語、BASIC、etc.充実した内容で、特にC言語を修得しようとしている僕にとっては、Cのコーナーはありがたい。GTフォーラムでは自分が疑問に思っていたことの答えがあったりして、今の自分には欠かせないものです。(大阪府/早乙女明・?歳)

▲「好き好き」「助けて」などと入力

▲音を出すときも画面上部に色帯が現れる

SYNTHE-VOICE?の使い方

■準備　RUNすると、ボイスデータを読みこみ、それを各配列におさめていく。そのあいだ、VRAMを介してやりとりしているので画面上部で細い色帯がチラチラする。

■入力　準備が終わったらSCREEN0にもどって、入力待ち状態になる。なるべく、あ行〜た行でできている言葉をローマ字で入力して(大文字、小文字は問わない)、終わったらリターンキーを押す。あ行〜た行以外はたんに無視する。

■発声　入力した文字列に従ってVRAMにデータを展開し(このため、画面上部にはまた色帯が現れる)、そのデータをPCMで鳴らす。

■くりかえし　発声が終わったら、スペースキーの入力待ちになる。スペースキーを押すと「入力」にもどる。

■リスト2 SYNTHE-VOICE? MSX turbo R 専用

```
100 '<<< synthe-voice? >>>
110 CLEAR 200,&HD000:DEFINT A-Z
120 N=2+128¥2
130 DIM D(N),A1(N),A2(N),I1(N),I2(N),U1(
N),U2(N),E1(N),E2(N),O1(N),O2(N),K(N),S(
N),T(N)
140 D$="$aAiIuUeEoOkKsStT"
150 '*** data-setting
160 COLOR 15,4,7:SCREEN 8
170 D(0)=128:D(1)=1
180 GOTO 320
190   '* sub [data-reading]
200 FOR I=0 TO 1
210 READ A$
220   FOR J=0 TO 31
230     L$=MID$(A$,J*4+1,2)
240     H$=MID$(A$,J*4+3,2)
250     A=2+I*32+J
260     B=VAL("&H"+H$+L$)
270     D(A)=B
280   NEXT
290 NEXT
300 COPY D TO (0,0)
310 RETURN
320   '* data-copying
330 GOSUB 190:COPY (0,0)-(127,0) TO A1
340 GOSUB 190:COPY (0,0)-(127,0) TO A2
350 GOSUB 190:COPY (0,0)-(127,0) TO I1
360 GOSUB 190:COPY (0,0)-(127,0) TO I2
370 GOSUB 190:COPY (0,0)-(127,0) TO U1
380 GOSUB 190:COPY (0,0)-(127,0) TO U2
390 GOSUB 190:COPY (0,0)-(127,0) TO E1
400 GOSUB 190:COPY (0,0)-(127,0) TO E2
410 GOSUB 190:COPY (0,0)-(127,0) TO O1
420 GOSUB 190:COPY (0,0)-(127,0) TO O2
430 GOSUB 190:COPY (0,0)-(127,0) TO K
440 GOSUB 190:COPY (0,0)-(127,0) TO S
450 GOSUB 190:COPY (0,0)-(127,0) TO T
500 '
510 '*** what to speak?
520 SCREEN 0
530 '* input speaking data
540 V=1:X=0:Y=0:A$=""
550 INPUT A$:L=LEN(A$):IF L=0 THEN 550
560 '* writing voice data to VRAM
570 SCREEN 8
580 FOR I=1 TO L
590 A=INSTR(D$,MID$(A$,I,1))¥2
600 IF A=0 THEN NEXT:GOTO 660
610 A=A*2-V:IF A<=10 THEN V=1 ELSE V=0
620 ON A GOTO 720,730,740,750,760,770,78
0,790,800,810,820,820,830,830,840,840
630 IF V=1 THEN Y=Y+1
640 NEXT
650 AD=Y*256+X-1
660 '* speak(?)
670 CALL PCMPLAY(@0,AD,3,S)
680 IF STRIG(0) THEN ELSE 680
690 IF STRIG(0) THEN 690
700 IF INKEY$="" THEN 510 ELSE 700
710 '*** sub [voice setting]
720 COPY A1 TO (X,Y):GOSUB 850
730 COPY A2 TO (X,Y):GOSUB 850:GOTO 630
740 COPY I1 TO (X,Y):GOSUB 850
750 COPY I2 TO (X,Y):GOSUB 850:GOTO 630
760 COPY U1 TO (X,Y):GOSUB 850
770 COPY U2 TO (X,Y):GOSUB 850:GOTO 630
780 COPY E1 TO (X,Y):GOSUB 850
790 COPY E2 TO (X,Y):GOSUB 850:GOTO 630
800 COPY O1 TO (X,Y):GOSUB 850
810 COPY O2 TO (X,Y):GOSUB 850:GOTO 630
820 COPY K  TO (X,Y):GOSUB 850:GOTO 630
830 COPY S  TO (X,Y):GOSUB 850:GOTO 630
840 COPY T  TO (X,Y):GOSUB 850:GOTO 630
850 X=-128*(X=0):Y=Y-(X=0):RETURN
1000 '*** voice data
1010 'a1
1020 DATA E50202FCFF020EDAAC829C3C26B6AF
244BBA9753649AD4DDB7E8FF0202E2FF0202E7AA
53A55902A1E22A2CB6924E708EAFCBB5DAFF6E02
55FF24029DC25EA073
1030 DATA 0287FF470294A64A5883ADE2BBB4FF
FF0202FFF10205F48B73A82018DEA1024AC6875C
708ED8D596E6FF73022FFF360291A74EB27B027E
FF4A0292B5595370AE
1040 'a2
1050 DATA EAAF9CFFFF0202B7FF0202BE8E50C5
5102A7FF290BB2AC565980C1DA93AFFFFF0202DF
FF0202C6935CD24502B5FF2902A6AB5D567AB6D1
A1BDFFFF020295FF02
1060 DATA 02BA9658D95202AFFF320296A35D63
729FD6B0B0FFFF640211FF1C0274B067D874028C
FF46026EA4706A6289DFCBA1FFFFEC0202FF7902
0AD182B5900236FF73
1070 'i1
1080 DATA 0933658BA7A0907D6C7D95C0D8E1E4
B584370A01012148799CABA789767282AFC9FDF1
E3AB6B320101012B688EADAE9A80687084B6C9F8
F9D09846130101103A
1090 DATA 779FBBB395806A7B99C7DEECF6BF86
30010101164077A7B7B99880717EA6D0F7F0F1B8
7C2B010101124A7BABB1AE91787180B2D4FEFDEE
B16B28010101155880
1100 'i2
1110 DATA A8AF9F856D6B80B8CBFEFEE0A9571C
0101012B7098BDB99880676B89BDD0F6FED2953A
0401010B3675ABC4BF9A806E7B9ACCF1F2FBBD7B
23010101134780ABB1
1120 DATA B2907C7787BBD2FEFDE5AB62250101
0122187A5A09482707A91C6DBF2F4BE862F01010
1194480ABB6B191807A8BB5CEFEF8E7A96426010
101225F86A19E93800
1130 'u1
1140 DATA 0A173973A7C8D6CE9E6E451E162E59
93DAFFFFFFFFB162110202020560B4E8FFFCBC80
3F0502205BA9FFFFFFFFEC8C2302020202319BE2
FFFFDA944907021556
1150 DATA A6FFFFFFFFF2800602020202040A1F6
FFFFD9883102020A52B7FFFFFFFFF87902020202
0239A9FFFFFFD88223020202042B7FFFFFFFFFF85
0202020202089CFFFF
1160 'u2
1170 DATA FFE97B0E0202022EACFFFFFFFFFF9D
0202020202020280FBFFFFFF8D0E0202020282FFFF
FFFFFFC30202020202020260E7FFFFFF9B1C020202
0263FFFFFFFFFFE11E
1180 DATA 02020202021AC3FFFFFFB430020202
0246FAFFFFFFFFFD3802020202020202AEFFFFFFBE
3802020222CEEFFFFFFFFFF420202020202020202A2
FFFFFFC1320202020202
1190 'e1
1200 DATA C36C032A1F8FA6DB84882E6D67B192
933A59416D97BAE5D4CF605E065947A6A0C1686F
436967857A796A769AD2FEFF714501274ECCCECF
6C4439627D92766356
1210 DATA 7AAEFAFFEF4B21012D6CD4DEBB692C
465E8D8D7954636AB2E6FFFB773D01243DBDC4DC
65583978828A5A573B6DA7E1FFFFC42409013780
CED3AC74475B629063
1220 'e2
1230 DATA 66396771D2F3FFFF9323010221B2DC
F583712B6D5C826167557A9ED9FFFFD52B010113
80E3F0B574304E58807A6D525D80BDFFFFFF6C31
011224CADBFC6A5B23
1240 DATA 6D7A876B584560A9D3FFFFF41D1401
215CE6E5DE6A403951867A66485F68C5E7FFFFD8
1301010986E9F8BB79224D428A786D4F6679B9F7
FFFFBD28010101A9F0
1250 'o1
1260 DATA 01016DBFC4A6896C48354F8CB39656
37486C8EB2D8F5FBDA6E01011888C1BA9E805C3C
324F84A6996E525E8BC5FBFFFF9C01010167E7FF
C4744027204080B2A9
1270 DATA 7A5C5E78ACFAFFFFA101010173F9FF
C4763B140C3680A99972697B8EB3FCFFFF870101
019CFFF5AD6C2F0101479DB98A504F7CA3CFFFFF
FF6F010101BCFFF49C
1280 'o2
1290 DATA 5C2501074DA1BA8E53486692D3FFFF
FF6E010101ABFFF5AA6521010147A4C08C504763
8ACCFFFFFF830101010199FFFCB56C2301013CA6C6
8C4A4C6F89B8FFFFFF
1300 DATA BA01010180FFFFC07737010123 90C5
934B4A7087ADFFFFFFE20101013EF7FFD9874805
010C7DC09B554B657191EEFFFFFF63010101A8FF
FABC7B230101459DA9
1310 'k
1320 DATA FF02FF02E4223AFF02C9021AA456CD
52FF02D19AFFFF62FF026E03C6FFB2FF3DFB0E69
450FAA2182025E02027D02ED15FFCFD58E68C650
F8F2FFFF90AA225A17
1330 DATA 2C830262020202020735BC4F6E0FFFF
FFFFFFFFFFFFFFFFF020202020202020202020713
1C262E363D43484E80FF343C4349D5E9363EFFFF
FFFFFFFFF11FFFFFFFF
1340 's
1350 DATA 89797C7B7C767A88878084646C7B82
92879B838A798375838 48D8480737677 7B7A0002
8583E7D7A78000282797679797483840002757A7
D70777B7C86797A7A
1360 DATA 839EA8A09C9EABC5D4C48D4C363D53
7682807B6467626E82A2AFC2C9E3FFFFA0010101
408D8082664B35222842778C8083A8DDFFFFFF01
0101409 28B6A8E5832
1370 't
1380 DATA FFFFFFFFB80501010101010101010101
3B708E968C806A57564E5A6F6D7291CFFFFFFFF7
C9B0948CA2989D99999E91797567594 3405E88B0
BDB1AA8F7871697472
1390 DATA 6E66748492A5A69B8E64554B3E4E5F
748FA4B5BFBAD1C9B6AA9183877974776F6B6658
5C645762726E7A82949D8F949CA1AAA1ACB39F8A
550F0B2931444B4F6A
```

110 メモリ初期化／整数型宣言
120 使用するコピー用配列の大きさNの計算
130 配列宣言
D(n) 音のデータをグラフィック画面に展開するための橋渡し的役割の配列
A1(n)〜T(n) 音「a」〜「t」のデータ。母音は前半と後半にわけてあるため、A1(n)、A2(n)のように2つある
140 入力チェック用文字列D$のセット
150 ■データセッティング
160 画面モード初期化(スクリーン8)
170 配列Dをグラフィックコピー用配列にするためのデータセット(図2参照)
180 行320へ飛ぶ
190 ■サブルーチン[データ読出]
200 Iのループ＝1回で1行ぶんのデータ
210 データ読み出し
220 Jのループ＝1回で2バイトのデータ
230〜240 データの長い文字列を4文字ずつ区切り、整数型の2バイトに変換するため前半2文字と後半2文字にわける
L$ 前半2文字(整数型の下位バイト)
H$ 後半2文字(整数型の上位バイト)
250 データを収める配列Dの添字Aの計算
260 収めるデータBの計算(整数型)
270 配列Dにデータを代入
280〜290 J、Iのループ閉じ
300 配列Dをスクリーン8の(0,0)にコピー(音声データを配列Dを通してVRAMに転送)
310 呼び出し先にもどる
320 ■各配列に音声データを転送
330〜450 サブルーチン(行190〜)でVRAMに展開したデータを配列A1〜Tに転送
510 ■入力受け付け
520 画面モード初期化(スクリーン0)
530 ■発声データ入力
540 新しい入力のたびの初期化
V 子音のあとの母音か、単独の母音か(0＝子音のあとの母音、1＝単独の母音)
X、Y コピー先座標
A$ 入力された文字列
550 文字入力待ち／入力文字列の長さを調べ、0(つまり無入力)なら行550へもどる
560 ■音声データのVRAM書きこみ
570 画面モード初期化(スクリーン8)
580 Iのループ＝入力された文字を1字ずつ評価
590 その文字が入力チェック用文字列D$のなかのどの文字かのチェック
A 文字判別コード。0は該当文字なし、1〜8が「a」〜「t」に対応
600 該当文字がなければ次の文字の評価へ
610 変数Aを飛び先指定用変数に変換／Aが10以下(母音)ならV＝1、それ以外はV＝0
620 Aに応じて分岐
630 単独母音(配列2個使用)なら座標変数Yを1増やす(1ライン下へ)
640 Iのループ閉じ
650 コピーした範囲のアドレス終値の計算
660 ■発声(?)
670 音声データをコピーしたVRAMアドレスの開始値(0)から終値(AD)までをPCMデータとして再生
680〜700 スペースキー待ち／キーバッファクリア
710 ■サブ[音声セット]
720〜840 音声データを座標(X, Y)へコピーする
850 書きこみ座標の更新／もとにもどる
1000 ■声データ
1010〜1390 「あ」〜「お」、k、s、tの音声データ ※母音のみ2行ぶん

Mファンにいいたい放題！

★マウスパッド欲しさに定期購読してしまった。しかし、毎号きちんと本屋さんで買ってたんだよ。だから、本屋で一冊売れ残るんじゃないかと心配。あ、それともそのぶん読者が一人増えるのかも。(埼玉県／今泉淳子・30歳)

BASICテクニック

人類の祖先はアフリカのマリアだそうだが、では生命の祖先はアメーバだ。アメーバをデジタルで作りだして人工生命問題への足掛かりとしたい。

→P85の「スーパー付録ディスクの使い方」参照

ギザギザのアメーバ

生命活動というのは、それ自体が、ひじょうにコンピュータ的な面を持っている。

そもそも、生命活動の背後で気が遠くなるほどの長い時間、連綿として形を保っているDNAというもの自体、まったくデジタル情報そのものなのだ。

一般に生物というと、なんとなくアナログな感じがするが、DNAは完全にデジタルな情報体だ。4種類の塩基が単純な規則を守って連なることで、アミノ酸の組み合わせを決定し、それがその生物のたんぱく質を決定する。たんぱく質の合成は、深遠な生命活動の底の底でおこなわれている現象にはちがいないが、じつは4種類の塩基の並び方で20種類のアミノ酸の組み合わせを決めているにすぎない。

この仕組みは、植物、動物を含めて、全生命のDNAに共通の仕組みだ。つまり、地球上の生命のDNAであるかぎり、データには互換性がある。フォーマットがおなじだから。

DNAは、デジタルだから、劣化しない。生命の誕生から見ていけば、現在のわたしたちのDNAは孫の孫の孫の……孫のコピーのはずだが、情報はまったく劣化していない。

†

そうした、きわめてデジタルなDNAが背後にいるのならば、生命活動というものをデジタルにおとしこめるのではないか。

少なくとも、プログラムで骨格を組んだアメーバくらいなら可能ではないか、というのが今回の計画の出発点だった。

†

単純な生命活動を箇条書きにすると、次の2つになる。

①現状維持
②増殖

現状維持とひとことでいっても、じつは、そのまえに、セルフアイデンティティというものが必要になる。つまり、生命にとって「現状」とはなにか。

抽象的に考えていると、どこまでいっても終わらないので、アメーバに限定しよう。

アメーバにとっての、セルフアイディンティティは、乱暴にいえば「自分はどこまで広がっているか」ということである。

アメーバにとって、世界は2つに大別される。1つは自分、もう1つは自分でないもの。自分と自分でないもののあいだには境界線があり、その境界線は連続してつながって、閉じた曲線になっていなくてはならない。

アメーバは、その境界線がずっと連続してつながっているように活動していく。

しかし、ただ活動しているだけではエネルギーを放出するだけで、エネルギー不変の法則から一方的にしぼんでいくことになるので、なんらかの形でエネルギーを取り込まなくてはいけない。

ある種のアメーバは、自分ではない別の物質を取り込んで、それを自分の体の一部に変化させ（消化・吸収）、変化しきれな

↑「お〜い」といって呼び出したアメーバの10分間にわたる生命活動の軌跡

↑本文の最後に書いてある改造を施すとこんなアメーバが動く

かった自分でないものを吐き出す（排泄）といった行動をとるだろう。また、ある種のアメーバは光をエネルギーに変換しながら活動を続けていくだろう。

そうして、そのあいだにほんの少しずつ、現状維持に失敗していく。小さな失敗がだんだん積み重なり、それが致命的な大きさにまで達したときそのアメーバは寿命を迎えることになる。

†

これが1つのアメーバのループである。このままだとあとはOkとカーソルが表示されているだけ、ということになりそうだが、もう1つ、アメーバは無限ループになる仕組みを持っている。それが、②の増殖だ。

親のループが始まって終わるまでのあいだに、子どものループが始まる。親と子どもは似たような性能を持っているので、寿命も似たようなものだ。したがって、子どものほうが親よりも長生きすることになる。これをくりかえすと、あるDNAグループが無限に生き延びていくことになるわけだ。

じっさい、人類を含め、種の保存に成功している生命は、けっきょくその種の源流にいたDNAが無限ループしているようなものなのだ。

†

したがって、①、②の2つをプログラム上で実現すれば、かなりりっぱなアメーバができるだろうというのが当初の目標だった。スタート時点で、数種類の遺伝子を用意し、その複数の遺伝子がたがいに捕食活動をすることで、自然淘汰されて、最終的によい種ができあがるのではないか。しかも、生命活動なんて、整理すれば、けっこうかんたんな仕組みじゃん……などとわたしは思った。

しかし、神の領域に足を踏み入れようとしたわたしに天罰はくだった。最終的にできあがったのは、人工生命どころか、「アメーバ泳動プログラム」ていどの形容しかできないものだった。神よ、許したまえ。

†

AMOEBA．BPZを走らせると、Please call me／とメッセージ出るので、すなおに呼びましょう。「おーい、アメーバくーん」でもいいし、パンパンと手をたたいてもいい。

すると、アメーバが生まれ、その後もずっと音にゆるやかに反応したり、静寂のゆらぎを吸収したりして、むよむよむよむよ、まおまおまおまお、ぷやぷやぷや……と動いていく。

付録ディスクに入っているプログラムは、左ページの左の写真のように骨組みの見えるアメーバだが、行460と行720をREM文にして、

```
475 PAINT (OX, OY), C, C
735 PAINT (OX, OY), C, C
```

の2行を追加すると、そのとなりの写真のようにピンクのアメーバになります。

↑呼び出すときにやさしい声でなく、手をたたいて呼び出すといきなりこんなアメーバになる

AMOEBA．BPZのプログラム解説 MSX turbo R 専用

100　■初期設定
110〜130　メモリ確保／整数型宣言／画面初期設定／ページ1を掃除
140　■アメーバの性格
150〜180　各種変数設定
BD　ボディを支えるスポークの数
C　外形線の色
OC　核の色
OX、OY　中心の初期位置座標
GT　最低限確保される大きさ
YR　アメーバの人生のひと区切り
AG　一定期間ごとにアメーバはAGぶん老化する
P　VRAMのページ切り換え用
OF　アメーバ全体が動くときの歩幅
DR　最長スポークの番号
ML　最長のスポークの長さ
HL　X方向の最小限界
HR　X方向の最大限界
VU　Y方向の最小限界
VD　Y方向の最大限界
AD　PCM録音の開始アドレス
200　■配列
210　配列宣言
ZS(n)、ZC(n)　スポークnの角度に応じた三角関数値
L(n)　スポークnの長さ
220　ZDの計算
ZD　スポーク1本あたりの角度
230〜260　各スポークの角度に応じた三角関数値をあらかじめ計算しておき、配列ZS、ZCに入れておく。
270　■スプライトの設定
280〜310　ページ0で順にスプライトの初期化、スプライトパターン番号0（アメーバの核になる）のパターン設定をおこなう
320　■誕生以前
330　グラフィック画面をオープンして文字がかけるようにする
340　座標(60，100)にLP(最終参照点)を設定する
350　"Please call me!"とメッセージ
360　スポークの数に相当するバイト数だけPCM録音
370　システム変数TIMEの初期化(ゴミ。TIMEでアメーバの寿命を管理しようと思っていたときの名残)
380　ディスプレイページとアクティブページを切り換え、画面クリア
390　■アメーバの最初の体を作る
400　スポーク0の長さを計算(PCM録音したデータを引き継ぐ)
410　スポーク0の先端にC色の点を置く
420　●各スポークの先端を結んでいくループ開始
430　そのスポークの長さをPCMデータ(行360で録音)によって計算
440　スポークの先端の座標計算
450　先端を線で結ぶ
460　スポークを描く
470　●ループ閉じ
480　核を表示
490　■アメーバの人生
500　●ループ開始
510　ディスプレイページとアクティブページの切り換え／アクティブページの画面消去
520　スポークの数の2倍のバイト数だけPCM録音
530　乱数RDの計算
RD　録音されたPCMデータの何バイト目からをスポークの長さデータとして使うかの位置決め用
540　●ループ開始
550　Dの計算(PCMデータから127を引いた値：PCMの127というデータは振幅の中心値)
560　スポークの長さの増減値A計算
570　スポークの長さの増減
580　増減した結果、長さが3未満なら5に設定
590　おなじく最低限の大きさよりも小さかったら、現状の値に3を加える
600　●ループ閉じ
610　スポーク0の先端にCのドットを置く
620　MLを初期化
630　●ループ開始
640　そのスポークの長さがML(最長)より長ければ、そのときのスポークの長さをMLにし、最長スポークの向いている方向をDRに入れる
660　スポークの先端の新しい座標X、Y計算
670〜700　X、Y方向の限界点を調べて、超えていたらそれぞれの処理のサブルーチンへ飛ぶ(X方向なら行840、Y方向なら行900)
710　スポークの先端を線で結ぶ
720　スポークを描く
730　●ループ閉じ
740　核を表示
750　新しい核の座標の計算
760　最長スポークからアメーバの歩幅ぶんを引く
770　行520で入力したPCMデータの再現
780　●ループ閉じ
790　■老化
800〜820　全スポークをAGぶん短くする
830　行490へ飛ぶ
840　■X方向の限界処理サブ
850〜890　限界を超えたぶんを切り取ってY座標とスポークの長さを変更してもどる
900　■Y方向の限界処理サブ
910〜950　限界を超えたぶんを切り取ってX座標とスポークの長さを変更してもどる

Mファンにいいたい放題！

★10月号のBASICピクニックのPCMはおもしろかった。ただ、リストを打ち込むのがちょっとめんどうだ。付録ディスクがあるのだから、本誌に掲載してあるプログラムはできるだけ収録するようにしてほしい。（山梨県／ミスターどん・19歳）★BASICピクニックについて。あまりみんなが使わないSCREEN3とSCREEN6のかしこい利用法を紹介してください（このまま誰にも使われなかったら哀れだ。あと5冊大変でしょうが、お体に気をつけてがんばってください（あぁ……悔しい……終わってしまうなんて）。（三重県／久世浩史・?歳）

BASICテクニック

→P84の「スーパー付録ディスクの使い方」参照

CGといえば3D、3Dといえばポリゴンと相場が決まっている。ポリゴンとはいったいなんなのか。家庭のBASICでポリゴンの仕組みを探る、その1。

ワイヤフレームの鉛筆

ポリゴン(polygon)のポリはポリエチレンのポリ(多)、ゴンはペンタゴン（5角形）のゴン(n角形)。あわせて、多角形という意味になる。多角形の組み合わせで立体を表現するCGテクニックの1つだ。

立体的CGにおけるポリゴンは、平面的CGにおけるドットによく似ている。ドットが細かいと緻密(ちみつ)なCGになり、ポリゴン数が多くなると表面が「なめらか」なCGに近づく。

立体をCGで表現する場合、まず大きくわけて2つの問題がある。1つは、立体データの持ち方（内部的問題)、もう1つは、いかに3Dっぽく見せるか(外見的問題）である。

♆

前者の内部的問題は、さらに3つにわかれる。

1つは、点と線だけで立体データを構成する「ワイヤフレーム（針金の枠）モデル」。2つめは、表面だけで立体を構成する「サーフェス（表面）モデル」。3つめは、中身までデータを持つ「ソリッドモデル」だ。ポリゴンは、このうちのサーフェスモデルの手法の1つなのだ。

ワイヤフレームモデルはデータとしては点と線だけだが、できあがったものは結果的に、向こうが透けてみえる多面体になる。多面体はようするに多角形の組み合わせだから、ワイヤフレームモデルとポリゴンは見た目がかなり近い。

すると、面が透明なポリゴンとワイヤフレームモデルとはまったくおなじだろうか。

たぶん、ちょっと違う。

今回は、このちょっとした違いを考えてみたい。

なお、外見的問題のほうはいろんなやり方がありそうだが、今回はなかでも比較的本格的な感じがするわりには計算が単純ですむ、「透視図法」での表示を試みた。これは、視点とスクリーンを決めておき、ある点を表示する場合、その点と視点とを結ぶ直線がスクリーンと交わる点をディスプレイ上にセットするというやり方だ。じっさいの物の見え方の原理とはちょっと違うが、見た目は自然の遠近感に近いといわれる。

♆

今回の付録ディスクに収録してある「3D−PEN. BP2」には数か所バグがあるので、右ページのプログラム解説の最初に書いてあるとおりに修正してほしい（誌面に掲載しているリストが最終バージョンのプログラムリスト。また操作方法も解説に書いてある。ようは遠近感のある鉛筆の線画をカーソルキーなどでクリクリと回して見るプログラムだ)。

ポリゴンは、すべて三角形に分割したほうが、あとの処理がしやすいので、鉛筆の底面も6つの二等辺三角形に分割した。

見た目はワイヤフレームだが、このプログラムのポリゴン的な部分は、鉛筆の表面を分割したすべての三角形の3頂点を2次元の配列Pで管理しているという点だ。

鉛筆のすべての面が三角形になっているので、描画ルーチンは配列Pで管理する三角形を描くくりかえしになる。行1460〜1510がそこだ。

以前、このコーナーでやったワイヤフレームモデルでは、立

■透視図法の座標計算

$$yy = y\frac{(EP-SC)}{(EP-z)}$$

↑透明ポリゴンによるちびた鉛筆

体を構成する辺をやはり１つの配列で管理して、すべての辺を描いていくことで立体を表示したが、それとよく似ている。

しかし、面を塗ろうとするときに決定的な違いがわかる。

ワイヤフレームモデルでは、面を塗ろうとしても、面の認識そのものがないので、いったいどこにＰＡＩＮＴの座標を設定すればいいかすらわからないが、このプログラムの場合は、三角形を描きながら、３つの頂点の中心点を計算して、三角形を描き終わった瞬間にその中心点でＰＡＩＮＴを実行すればいい。

面が透明なポリゴンとワイヤフレームモデルとの「ちょっとした違い」はまさにこの点で、きちんとしたポリゴンCGになるための重要な構造の違いを持っているのである。

では、なぜ、今回のプログラムは面を塗っていないのか？

じつは、ポリゴンの最大の特質は、各面を「向こう」側から描いて塗っていくことで「こちら」側から見えないはずの面はCG上でも見えなくなるという点なのだ。ワイヤフレームで、もしそういうことをやろうとすれば、考えられないほど複雑な処理になりそうだが、ポリゴンはたんに「向こう側から塗っていく」だけでいい。

だが、しかし／　いったい、「向こう」というのをどう判定すればいいのだろう？　これが、じつは最大の謎なのである。以下次号。　　　（コルサコフ）

「3D－PEN. BP2」の解説

■修正点

付録ディスクに収録してあるプログラムと掲載リストとは、以下の行が異なります。掲載リストのほうが正しいので、行1285を追加し、行1290～1320、1510、1560、1570を掲載リストどおりに修正してください。

■操作方法

カーソルキー左右　Ｙ軸中心の回転
カーソルキー上下　Ｘ軸中心の回転
ＩＮＳ、ＤＥＬ　Ｚ軸中心の回転
「１」「２」　視点（ＥＰ）の前方・後方移動
「－」「￥」　スクリーン（ＳＣ）の前方・後方移動
※キー入力を受け付けると１回ＢＥＥＰ音を鳴らして描画する。

■プログラムで使用した主要変数

ＫＮ、ＫＹ、ＲＶ　キー入力受け付け用。ＲＶは－１か１で、回転方向の順、逆を管理
ＯＸ、ＯＹ　３次元座標系の原点の画面表示上の座標
Ｐ、Ｑ　画面ページ切り換え用
Ｐ(ｎ，ｍ)　ポリゴンｎを構成する頂点番号
Ｒ　画面比率調整用
ＳＣ、ＥＰ、Ｄ　ＳＣはスクリーンのＺ座標、ＥＰは視点のＺ座標。Ｄは、スクリーンと視点の位置調整用増分
Ｔ(ｎ，ｍ)、Ｃ(ｎ)、Ｕ　回転移動計算用。Ｕは20分のπで回転するときの単位角度としている
Ｘ(ｎ)、Ｙ(ｎ)、Ｚ(ｎ)　３次元座標系での各頂点のＸＹＺ座標
ＸＸ(ｎ)、ＹＹ(ｎ)　スクリーン上(Ｚ平面に平行）のＸＹ座標

■プログラムの構成

1000～1070　初期設定
1080～1170　データ読みこみほか
1180～1270　キー入力
1280～1330　ズームイン／アウト
1340～1390　回転計算
1400～1530　画面表示
1540～1570　キー入力用データ
1580～1590　頂点の数、ポリゴンの数、原点の画面表示上のＸＹ座標
1600～1820　各頂点のＸＹＺ座標
1830～2210　各ポリゴンのデータ（三角形を構成する点の番号）

サンプルプログラム「3D－PEN. BP2」 MSX2/2+ VRAM128K

```
1000 '<<< 3D-pencil >>>
1010 '=== first setting
1020 DEFSNG R-Z:DEFINT A-Q,S
1030 COLOR 15,0,4:SCREEN 7
1040 P=1:Q=0:SETPAGE ,1:SCREEN,0
1050 R=1.7:SC=50:EP=100:D=5
1060 U=ATN(1)/6
    :DIM T(2,2):T(0,0)=1:T(1,1)=COS(U)
    : T(2,2)=T(1,1):T(2,1)=SIN(U)
    : T(1,2)=-T(2,1)
1070 DIM C(5)
    :FOR I=0 TO 1
    :  FOR J=0 TO 2
    :    C(J+I*3)=J
    :  NEXT
    :NEXT
1080 '=== read data/dimension arrays
1090 READ KN
1100 FOR I=0 TO KN:READ KY(I):NEXT
1110 READ N,M,OX,OY
1120 DIM X(N),Y(N),Z(N),XX(N),YY(N)
1130 DIM P(M,3)
1140 'first position
1150 FOR I=0 TO N
    :  READ X(I),Y(I),Z(I)
    :NEXT
1160 FOR I=0 TO M
      :FOR J=0 TO 3
      :  READ P(I,J)
      :NEXT
    :NEXT
1170 GOTO 1400
1180 '=== input
1190 KY=ASC(INPUT$(1))
1200 IF RV<0 THEN SWAP T(1,2),T(2,1)
1210 RV=1
1220 FOR A=0 TO KN
      :IF KY=KY(A) THEN BEEP:GOTO 1250
1230 NEXT
1240 GOTO 1190
1250 IF A MOD 2 THEN SWAP T(1,2),T(2,1)
                         :RV=-1
1260 A=A¥2
1270 IF A<3 THEN 1340
1280 'zoom in/out
1285 S=SC:E=EP
1290 IF A=3 THEN S=SC-D*RV
1300 IF A=4 THEN E=EP-D*RV
1310 IF E<S OR E<80 OR E>200 OR S<-20
      THEN BEEP:BEEP:GOTO 1180
1320 SC=S:EP=E
1330 GOTO 1400
1340 'calculate to rotate
1350 FOR I=0 TO N
1360   X=X(I)*T(C(A  ),C(A  ))
        +Y(I)*T(C(A  ),C(A+1))
        +Z(I)*T(C(A  ),C(A+2))
1370   Y=X(I)*T(C(A+1),C(A  ))
        +Y(I)*T(C(A+1),C(A+1))
        +Z(I)*T(C(A+1),C(A+2))
1380   Z=X(I)*T(C(A+2),C(A  ))
        +Y(I)*T(C(A+2),C(A+1))
        +Z(I)*T(C(A+2),C(A+2))
1390   X(I)=X
      :Y(I)=Y
      :Z(I)=Z
    :NEXT
1400 '=== display
1410 CLS
1420 FOR I=0 TO N
1430   XX(I)=X(I)*(EP-SC)/(EP-Z(I))*R+OX
1440   YY(I)=Y(I)*(EP-SC)/(EP-Z(I))+OY
1450 NEXT
1460 FOR I=0 TO M
1470   PSET (XX(P(I,2)),YY(P(I,2)))
1480   FOR J=0 TO 2
1490     LINE -(XX(P(I,J)),YY(P(I,J)))
1500   NEXT
1510 NEXT
1520 SETPAGE P,Q:SWAP P,Q
1530 GOTO 1180
1540 '=== data
1550 '    kn,ky (key assign)
1560 DATA 9
1570 DATA 30,31,127,18,28,29,94,92,49,50
1580 '     n  m  ox oy
1590 DATA 19,35,256,100
1600 'top (0)
1610 DATA  0    ,-54.64,  0
1620 'upper (1-12)
1630 DATA  0    ,-20   , 20
1640 DATA   8.66,-28.66, 15
1650 DATA  17.32,-20   , 10
1660 DATA  17.32,-28.66,  0
1670 DATA  17.32,-20   ,-10
1680 DATA   8.66,-28.66,-15
1690 DATA  0    ,-20   ,-20
1700 DATA  -8.66,-28.66,-15
1710 DATA -17.32,-20   ,-10
1720 DATA -17.32,-28.66,  0
1730 DATA -17.32,-20   , 10
1740 DATA  -8.66,-28.66, 15
1750 'lower (13-19)
1760 DATA  0    , 40   , 20
1770 DATA  17.32, 40   , 10
1780 DATA  17.32, 40   ,-10
1790 DATA  0    , 40   ,-20
1800 DATA -17.32, 40   ,-10
1810 DATA -17.32, 40   , 10
1820 DATA  0    , 40   ,  0
1830 'upper polygons (0-11)
1840 DATA  0, 1, 2, 1
1850 DATA  0, 2, 3, 2
1860 DATA  0, 3, 4, 2
1870 DATA  0, 4, 5, 3
1880 DATA  0, 5, 6, 3
1890 DATA  0, 6, 7, 4
1900 DATA  0, 7, 8, 4
1910 DATA  0, 8, 9, 3
1920 DATA  0, 9,10, 3
1930 DATA  0,10,11, 2
1940 DATA  0,11,12, 2
1950 DATA  0,12, 1, 1
1960 'side polygons (12-29)
1970 DATA  2, 1,13, 5
1980 DATA  2,13,14, 5
1990 DATA  2,14, 3, 5
2000 DATA  4, 3,14, 6
2010 DATA  4,14,15, 6
2020 DATA  4,15, 5, 6
2030 DATA  6, 5,15, 7
2040 DATA  6,15,16, 7
2050 DATA  6,16, 7, 7
2060 DATA  8, 7,16, 8
2070 DATA  8,16,17, 8
2080 DATA  8,17, 9, 8
2090 DATA 10, 9,17, 9
2100 DATA 10,17,18, 9
2110 DATA 10,18,11, 9
2120 DATA 12,11,18,10
2130 DATA 12,18,13,10
2140 DATA 12,13, 1,10
2150 'botom polygons (30-35)
2160 DATA 19,13,14, 3
2170 DATA 19,14,15, 3
2180 DATA 19,15,16, 3
2190 DATA 19,16,17, 3
2200 DATA 19,17,18, 3
2210 DATA 19,18,13, 3
```

Ｍファンにいいたい放題！

★私がＭＳＸを初めて触ったのは、今から6年前、まだＭＳＸが元気なときでした。その後、より活用させるためにＳＴを買い、いろいろと手を加えて環境を整えてきました（ユーザーサポートは受けられなくなりましたが）。ＳＴは自分が全部お金を出して買った初めてのパソコンであり、現在でも現役で活躍中です。ＭＳＸのソフトや情報誌が消えつつあるなかで、自機を活かすために、最近は今までよりも力を入れて勉強しなおしています。Ｍファンが終わるまで、現在の努力を無駄にしないためにも、ＭＳＸユーザーのためにせいいっぱいのサポートをよろしくお願いします。（栃木県／高木登志男・？歳）

BASICピクニック

前号からの続き。空間における「向こう」は、プログラムでどのように判定されるのか？ 今回は、その問いに対する1つの解答計画を示す。

ポリゴンのための空間座標の基礎知識

【前号までのあらすじ】

ちびた鉛筆をいくつかの三角形にわけて立体的に表示するプログラムを闇雲(とりあえず)に作ってみたものの、透視図法で線画を表示した時点で誌面が尽き、しめ切りにも見放されたBASICピクニック担当コルサコフ山島。

そもそも、この世界には、いつのころからか語り継がれてきた伝説があった。

「立体物をポリゴンの組み合わせで構成し、向こうのポリゴンから塗っていけばかんたんに陰線処理（見えないはずの稜線を消すこと）ができる」

人々は（わたしだけだが）そのことばだけを漠然と信じ、一夜にして「3D-PEN.BP2」を組み上げた。だが、しかし、三角形のポリゴンを向こうから塗りつぶそうとして、わたしはその伝説の真の意味に気づいた。

伝説は、じつは2つの要素に分けられていたのだ。

①向こうのポリゴンから塗っていけば

②かんたんに陰線処理ができる

芳香を発する食虫植物ウツボカズラの芳香、光でエサを呼び寄せるチョウチンアンコウの光に相当するのが②の部分で、わたしは、それにつられて寄っていき、①をまともに見た。そして、石になってしまった。

⚜

3D表示しようとすると、立体の形のバリエーションはいろいろな影響を与えてくるが、とりあえず、鉛筆のように、どの稜線を見ても凸になっている立体を考えてみよう。

鉛筆を見ている自分の視線を想像すると、鉛筆を構成する面には2つの様相しかないことがわかってくる。それは、

a1　見える面
a2　見えない面

の2つだ。

ところで、ワイヤフレーム状の立体ではすべての面が見えているわけだが、その状態で考えると、

b1　表が見える面（表向き）
b2　裏が見える面（裏向き）

の2つに大きく分かれている。

しばらく、ワイヤフレーム状になった鉛筆を頭のなかで想像して思いをめぐらすと、a1とb1、a2とb2はまったく一致することが直観的にわかるだろう。

■図1　2つのポリゴンと視点平面との関係

■図2　3点を通る平面の方程式

平面Q

Q1(a1, b1, c1)

Q2(a2, b2, c2)

Q3(a3, b3, c3)

平面の方程式

$$\{b1(c2-c3)+b2(c3-c1)+b3(c1-c2)\}X$$
$$+\{c1(a2-a3)+c2(a3-a1)+c3(a1-a2)\}Y$$
$$+\{a1(b2-b3)+a2(b3-b1)+a3(b1-b2)\}Z$$
$$+a1(b3c2-b2c3)+b1(c3a2-c2a3)+c1(a3b2-a2b3)=0$$

つまり、そのポリゴンが見えるか見えないか（隠されるか、隠されないか）は、視点から見て表向きか裏向きかで判定できるのだ。

さて、あるポリゴンの表裏の判定は「法線ベクトル」（その平面の方程式の係数を要素とするベクトルで、その平面と直交する）というものを使えば、かんたんらしい。法線ベクトルはその平面上の3点の座標（つまりポリゴンの座標）から計算できる（ただし、表裏の判定に使うには、あらかじめ法線ベクトルを計算する座標の順番をそろえておく必要がある）。

そして、法線ベクトルと視線のベクトル（視点からそのポリゴンの1点に伸びるベクトル）とのなす角度の絶対値が90度より大きいかどうかを判定すればいい。90度より大きければ表向き、小さければ裏向きだ。

このへんの計算は、さいわいなことに三角関数を使う必要もなく、四則演算だけでやりとおせることがわかった。

こうして、表向きのポリゴンだけをピックアップし、それだけを表示するというやり方をとれば、陰線処理も必要なく、きれいに鉛筆が表示できるのだ。

しかし、けっきょくその道はとらなかった。鉛筆しか表示できないのではつまらない。せめて、階段を表示できる方法がほしかったからだ。

階段をこの方法で表示しようとすると、うまくいかない。というのも、階段のような立体には、第3の「一部が隠されている面」というものが出てくるからだ。また、ある場合には裏向きのポリゴンを見せたいこともきっとあるだろう。

じゃあ、どうすればいいか。

⚜

わたしは、日夜、空中に浮かぶポリゴン物体のイメージを牛のように反芻しながら考えていたのだが、その長く苦しいフリーシンキングの果てにやっとたどりついたのが、たったの図1である。

視点EPから見た、隣りあう2枚のポリゴン（稜線Aを共有し、別々の頂点P1、P2を持つ）の関係は2種類しかない。

①重なる（どちらかがどちらかを隠す）

②重ならない（たがいに隠しあわない）

これはEPとAでできる平面αを壁として考えると、それぞれ次のようにいいかえられる。

①2枚ともおなじ側にある

②別々の側にある

ということは、2つのポリゴンが重なるか、重ならないかは、αを基準面にして考えれば判定できるのだ。つまり、点P1、P2と平面αとの位置関係を判定すればいい。

⚜

平面の方程式は、3点の座標があたえられれば、図2のように求められる。平面の方程式を$f(x,y,z)=0$とすると、直線とおなじように、次のような性質がある。

平面上の点を(a,b,c)とすると、

$f(a,b,c)=0$

※以下、式fに点Pの座標を代入したものをf(P)と表記する

平面上にない点P1、P2の場合は、f(P1)もf(P2)も正か負のどちらかになる。

そして、ここからが重要なのだが、その平面を基準にして点P1とP2がおなじ側にあると、f(P1)とf(P2)の符号はおなじ、逆に別々の側にあると、符号が異なるのだ。

ということは、

・視点EPと稜線Aでできる平面の方程式を$f(x,y,z)=0$

・ポリゴン1、2の稜線A上にない頂点をP1、P2

とすると、

①重なる場合

$f(P1)*f(P2)>0$

②重ならない場合

$f(P1)*f(P2)<0$

ということがわかる（図3）。

重なる場合、どちらが手前かを判定しなくてはならないが、これはいまの手法をくりかえすことで判定できる。

ポリゴン1を1つの基準平面として、EPとP2がおなじ側にあるかどうかを判定すればいいのだ。おなじ側にあれば、ポリゴン1のほうが向こう、異なる側にあればポリゴン2のほうが向こうにあることになる。

こうして、すべての稜線に関して隣りあうポリゴンの前後関係は完全に調べられる。

そして、あるポリゴンを表示するまえにそのポリゴンが隠す別のポリゴンの有無を判断し、あった場合はそのポリゴンがすでに表示されているかどうかを判定すればいいのだ。

これで計画は万全。あとはプログラムを組むだけだ。

次の号では、まったく別のテーマ（予定では人工知能）に体当たりするつもりだが、付録ディスクの片隅に（こんどこそ）今回の計画を実行にうつしたプログラムを収録する予定。

■図3　点と平面と式の値の関係

P2(x2, y2, z2)

P1(x1, y1, z1)

$f(x1, y1, z1)*f(x2, y2, z2)>0$

稜線Aと視点EPでできる平面α
$f(x, y, z)=0$

P2
(x2, y2, z2)

P1(x1, y1, z1)

$f(x1, y1, z1)$
$*f(x2, y2, z2)<0$

稜線Aと視点EPでできる平面α
$f(x, y, z)=0$

【参考文献】　「アスキーラーニングシステム／実習グラフィックス」（アスキー／税込み1850円）、「モノグラフ11／空間図形改訂版」（科学新興社／税込み620円）

BASICピクニック

番外編

パソコンブーム、次世代ゲーム機ブーム、阪神大震災、サリン事件、超円高、青島都知事……世の中は革命的にドサクサしている。このすきに自分で自分を語るウロボロス的番外篇。

BASICピクニックをピクニックする

ほんとうは、こういうページは最終回のオマケとしてやろうと思っていたのだが、非現実的なほど高性能なゲーム機が出回ったり、サリン事件で足止めをくらったり、1ドル80円になったり——ぼく（コ）や編集長が小学生のころは、1ドル360円だったのに——、大好きなコント作家兼役者だった青島幸男が都知事になったり、いま世の中はほんとうにすごい。前例とか歴史とか伝統とか習慣とか規則とか永遠とか、そういう壁を完全に突き抜けてしまって、時代はブレイクスルーだ。

そういうわけで、今回は、じつは3回でやめるつもりだった、このBASICピクニックという記事の企画と、その人生での応用について書いておきたい。

†

ぼくには娘が2人いて、上の子は最近小学生になった。BASICピクニックは、その上の子が生まれる半年前の春にはじめた企画だったと思う。

最初は3回しか考えていなかった。BASICでプログラムを組むということが、どのていどにかんたんで、どのくらいおもしろいことなのかを、幼稚園の遠足感覚で見せて、BASICの潜在需要を掘り起こすキャンペーン企画のつもりだった。

やりたいことは、最初3つしかなかったのだ。VRAMのカラーテーブルとパターンジェネレータテーブルと多色刷り。多色刷りとは、SCREEN1でキャラクタ1文字ごとに16色が使える有名な技術のことだ。

ぼく自身、当時のファンダム担当デスクとして、とくに多色刷りが不思議でおもしろくてしかたがなかった。それで、自分でBASICを勉強して、自分で理解したことを記事にしようと考えたわけだ。

真理のカギは、よくわかっていない状態から、よくわかる状態に変化していく途中に、きっと落ちている。それを記事にすれば、多くの人たちの疑問符がとれるはずだ。それがBASICピクニックをはじめた中心的なねらいだった。すくなくとも最初の3回までは、このねらいはピッタリあたったと思う。

しかし、3回ぶんが終わったあとにかなり本質的な悩みが残った。続けるべきか。続けるとすれば、なにをやればいいのか。BASICにくわしいわけではないぼくが、この記事を続けていく意味があるのか。

†

その本質的な疑問を抱えながら、こんなに長く続いた。Mファンがなくなるその日まで続くなんて、ほんとうは考えていなかったのに。

そのあいだ、すっかりぼくはBASICにくわしくなった。マシン語もちょっとかじりはじめたころ、Mファンにディスクが付き、おなじころ次女が生まれ、それをきっかけにぼくは40代の主婦なみの費用をかけて運転免許を取った。そして、最初に買った車は事故でつぶれ、次女は今年から幼稚園にあがった。

このあいだに、いつのまにか、当初の「BASICへの誘い」的なテーマから、もっぱら「ちょっとBASICのできるおじさんが、×××をやろうと思ったら……」というテーマに変わっていたのだ。いつ変わったのか、自分でも記憶にないが、しかし、このテーマは、前期BASICピクニックよりもテーマとしては広がりがある。

■図　BASICピクニックのフローチャート

- スタート
- テーマを考える
- やさしすぎる（YES→テーマを考える／NO）
- むずかしすぎる（YES→テーマを考える／NO）
- 資料を探す
- あった（NO→資料をあきらめる／YES→考える）
- 資料をあきらめる（NO→資料を探す／YES→考える）
- 考える
- めげる（YES→考える／NO）
- わかる（NO→考える／YES）
- プログラムを組む
- めげる（YES→未完成／NO→完成）
- 完成
- 未完成

ちょっとBASICのできるおじさんとは、ぼくのことだが、「おじさん」という表現には、一般的な家庭という意味をこめている。つまり、プログラミング技術や数学に特別くわしいわけではないし、プログラムを組むにあたって特別に相談できる専門家がいるわけではないという環境を前提にしているのだ。

その前提のうえで、後期BASICピクニックの諸テーマは考えられ、解決されてきた。FM音源の音色をエディットしたり、英単語帳を作ったり、ローンの計算をしたり（これは家を買うときに現実に使った）、8パズルを解いたり、立体図形を描いたり、ポリゴンを動かしたり（これはまだ完全には解決されていないが）……。きちんとしたプログラムとして評価しようとすると、多くの問題を抱えたものばかりだが、ともかく、MSXのBASICとせいぜい図書館で手に入る資料で、ふつうのおじさんがプログラムを組んできたのだ（たいていは資料がなく、ほとんど無手勝流でやってきた）。

家庭にMSXが1台あったら、BASICでこんなコンピュータっぽいことができる。どれ、おじさんがやってみせよう。う〜んう〜ん（悩んでいる）、ほ、ほら、できた（喜んでいる）。

そうした家庭内プログラマの姿が後期BASICピクニックのテーマであり、Mファンなきあとのあなたに提案する、1つのライフスタイルなのだ。（コ）

BASICピクニック

つねにテーマは残されていく。馬の鼻先にぶらさげられたニンジンのように、武蔵野の逃げ水のように、ドケチの永久保存版梅干しのように。

残されたテーマ、なんつって

わたし（コルサコフ）のけっして短くはない人生のうちで、こんなに力いっぱい考えたのははじめてだが、けっきょくできなかった。4月号のこのコーナーで堂々と展開してしまったポリゴン表示計画のことである。

法線ベクトルを使ったものなら、鉛筆の表示だけは確実にできたはずだが、くやしくてその方向には進めなかった。参考書に書いてあった一般的なポリゴンの表示は、むずかしくてまねできなかった。

それで、階段状のポリゴン立体も表示できるはずの、かんたんでオリジナルなポリゴン表示方法を思いついたつもりだったのだが。

もう一度、要約すると、

①稜線Aで隣りあうポリゴンP1とP2を考える。視点と稜線Aとで作る平面（視点平面と呼ぶ）を基準にして、P1とP2がおなじ側にあるかどうかを調べることで、2つのポリゴンが視点から見て重なるかどうかがわかる。

②重なる場合は、P1を基準平面として、P2のもう1つの頂点と視点とがおなじ側にあるかどうかを調べることで、P1とP2の前後関係もわかる。

③すべてのポリゴンについて、①と②をくりかえしてデータを取れば、視点から見たポリゴンどうしの全体的位置関係が判定できる。

④その位置関係に従って、向こう側からポリゴンを表示していけばきれいな正しい立体が表示できる。

†

①と②までは、さすがに数学的に単純な真理であるだけに、プログラムもそこまではできあがった。問題は③だった。

ある三角ポリゴンPと隣りあうポリゴンはa、b、cの3個ある。それについて、①と②の操作をくりかえすと、たとえば図1のような関係にあることがプログラム上で把握できる（ここではcはPの向こう側にある）。すべてのポリゴンについて、図1のような関係までは完全に調べがついた。

そこで、L（p，n）という配列を考えよう（pはポリゴン番号）。

L（p，0）～L（p，2）……ポリゴンpと隣りあうポリゴンのポリゴン番号

L（p，3～L（p，5）……上記の順に隣りあうポリゴンとの関係（たとえば、－1だとpが手前、1だとPが向こう、0だと隠しあわない）

この配列を、鉛筆を構成する30数個のポリゴンについて計算するとターボRでも10秒くらいはかかったが、まあそれはしかたがない。

この配列をいざ活用するという段になると、いちばん最初に考えた「あるポリゴンを表示しようとするまえに、そのポリゴンが隠す別のポリゴンの有無を判断し、あった場合はその隠されるポリゴンがすでに表示されているかどうかを判定する」という考え方は不十分だったことに気づいた。というのも、〝そのポリゴンが隠す別のポリゴンの有無〟がそうかんたんにはわからないからだ。あるポリゴンが隠すポリゴンは、かならずしも隣りにあるとはかぎらない。だから、配列Lを総合して判断しなくてはいけないのだが……いったいどうやって？

■図1　ポリゴン計画の最終到達図

最終的に、次のような結論に達した。ある立体を構成するポリゴンのうち、任意のポリゴンどうしの前後関係は、隣りあうポリゴンの前後関係の情報からは導き出せない（背理法で証明できる）。

ただ、任意のポリゴンどうしの前後関係がわからなければいけないわけでもない。じつは、配列Lによって、全ポリゴンをいくつかのレイアー（たがいに隠しあわず、隣りあうポリゴン群）にまとめ、レイアー単位でスクリーン上に多角形を投影し、その多角形の重なりや、重なった場合の前後関係を判定していけばどうにかなるのではないかとまだ思っている。

†

最後であるのをいいことにまったく話は変わるが、1つ、なんとなく気になっている現象がある。

それは、COPY文の合わせ鏡的な現象だ。27ページのリストを見てほしい。

ここで使われているループは、キー入力のための行140－210のGOTO文によるループだけ。そして、CIRCLE文もCOPY文も1回しか使っていない。

にもかかわらず、写真のように、カーソルキーの左右を押すたびに、複数の円が表示されるのだ。

参考プログラム　自己増殖COPYのサンプル

```
100 SCREEN 5:COLOR 15,0,4:CLS
110 U=ATN(1)/9:AN=9
120 R=20:D=10:C=15:CX=30:CY=30
125 DX=255-D:DY=211-D
130 GOTO 180
140 S=STICK(0)
150 IF NOT(S=3 OR S=7) THEN 140
160 A=AN+(S=3)-(S=7)
170 IF NOT(A>18 OR A<0) THEN AN=A:CLS
180 BEEP:CIRCLE (CX,CY),R,C
190 X=D*COS(AN*U):Y=D*SIN(AN*U)
200 COPY (0,0)-(DX,DY) TO (X,Y),,TPSET
210 GOTO 140
```

【プログラム解説】行110のATN(1)はアークタンジェントで、タンジェントの値が1になる角度（つまり45度）のラジアン値を計算。行140でカーソルキーの入力を受け付け、左右の場合のみ反応する。行170は画面外に出ないようにするための処理。行180で画面左上に円を1つ描き、行200で複写する（TPSET だと透明部分を複写しないので重なりが生きる）。このときのDX、DY、X、Yの値の取り方がポイント。

【変数の意味】U＝5度に相当するラジアン／AN＝複写方向の角度の単位数／R、C＝円の半径と色／CX、CY＝円の中心座標／D＝複写元と複写先の距離（ドット）／DX、DY＝複写範囲の右下座標／S＝カーソルキー入力

プログラムを走らせると、まず画面の左上に半径20ドットの円を1個だけ描く

その直後、一気に画面右下に向かってその円を連続複製する

カーソルキーの左右を押すたびに、

円の複写方向が5度単位で変化する

■図2　自己増殖COPYの原理

領域aを領域bに複写する

★	♡	○	△											

a　b

①

★	♡	○	△		★									

②

★	♡	○	△		★	♡								

③

★	♡	○	△		★	♡	○							

④

★	♡	○	△		★	♡	○	△						

⑤

★	♡	○	△		★	♡	○	△						

⑥

★	♡	○	△		★	♡	○	△		★				

⑦

★	♡	○	△		★	♡	○	△		★	♡			

領域aを領域bに複写する場合、一度に複写するのではなく、VRAM上で範囲の端から1ドットずつ複写していく。2つの領域が重なっていると、複写による変化が領域aにも影響し（⑥に注目）、自己増殖COPYとなる。

この複数の円は、行200のCOPY文が1回で作りだしている。

†

これは、複写する範囲と複写先の位置の取り方にポイントがある。円だけでなく、なにもない部分も含めて、ほぼ画面全体を複写し、さらに、複写先は複写元の範囲の広さとは反対に、ほんのちょっとだけずらした地点にするのだ。

すると、座標の片端から複写していく（一瞬のことだが）に従って、複写元の状況は変化していく。もともとなにもなかったところに、円が複写されてくるからだ。この複写元の変化は、まだ複写が完了していない段階で起きるので、次々に複写結果に影響していく。その結果、たくさんの円を画面中に表示することになるわけだ。

この現象は、図2のように直線的なサンプルにすると、わかりやすいだろう。

ここに掲載した参考プログラムでは右下方向にしか自己増殖しないが、COPYの範囲指定を左上ー右下ではなく、右下ー左上とか、右上ー左下などにすると、複写していく方向が変わり、複写元（複写開始地点）を適当にとってやると全方向に自己増殖COPYをする。いくつかの場合分けと変数管理を加えれば、1個のCOPY文で周囲に図形を放射するプログラムができるだろう。

†

もう1つ、人工知能もちゃんとやりたかった（といっても、AI風呂沸かしとか、ファジー炊飯とかのレベルの人工知能だが）。以前やった8パズル解答プログラムは、人工知能の1分野ではあるらしいが、もっとインタラクティブなAIをやりたかった。ファイル操作を使った、いわゆる“人工無能”と顔の表情変化を連動させて、パラメータしだいで怒った顔になったり、情けない顔になったりするものを、と考えていたが、多少言語学的にきちんとしたものをと考えているうちに、時間が来てしまった。関係ないが、その過程で読み返してふたたび魂をゆさぶられた本に、岩波新書『日本語の起源』（大野晋・著／旧版と新版の2種類ある）がある。

†

世界の進展は、脇道との戦いであり、しかも、脇道のどれかに最終解脱へと通じる道がある。MファンやBASICピクニックは、わたしにとっても、あなたにとっても、そういう意味で脇道の1つだった。そうして、1つの脇道が閉じるたびに世界はより前に進むのだ。　（コ）

Mファンにいいたい放題！

★MSXという媒体にこだわっている人も多いようですが、大切なのは「何かを創ろうとする人」（アマチュア）、「それらを楽しもうとする人」が共存できた媒体であるということではないでしょうか。晩年は第三者（前出の2者の次に重要という意味で）のソフトハウスさんが離れてしまい、Mファンも休刊となりましたが、ここまでMファンを発行し続けていただいた意義は想像以上に大きいと思います。それはともかく、「MSXの意志を受け継いだ媒体」が世に出る日もそう遠くないのではないでしょうか。そんな気がします（気のせいかも）。そのときは、世界中のアマチュアクリエイターさんのためにも復刊してくださいね。（兵庫県／冨岡優・18歳）